

# Zint Barcode Generator and Zint Barcode Studio User Manual

Version 2.10.0.9

May 2022



# Contents

<b>1. Introduction</b>	<b>5</b>
1.1 Glossary	5
<b>2. Installing Zint</b>	<b>7</b>
2.1 Linux	7
2.2 Microsoft Windows	7
2.3 Apple macOS	7
2.4 Zint Tcl Backend	7
<b>3. Using Zint Barcode Studio</b>	<b>8</b>
3.1 Main Window and Data Tab	8
3.2 Composite Groupbox	9
3.3 Additional ECI/Data Segments Groupbox	10
3.4 Symbology-specific Tab	11
3.5 Appearance Tab	12
3.6 Colour Dialog	13
3.7 Data Dialog	13
3.8 Sequence Dialog	14
3.9 Export Dialog	15
3.10 CLI Equivalent Dialog	15
<b>4. Using the Command Line</b>	<b>16</b>
4.1 Inputting Data	16
4.2 Directing Output	17
4.3 Selecting Barcode Type	17
4.4 Adjusting Height	19
4.5 Adjusting Whitespace	19
4.6 Adding Boundary Bars and Boxes	20
4.7 Using Colour	21
4.8 Rotating the Symbol	21
4.9 Adjusting Image Size	22
4.9.1 Scaling Example	22
4.9.2 MaxiCode Raster Scaling	22
4.10 Input Modes	23
4.10.1 Unicode, Data, and GS1 Modes	23
4.10.2 Input Modes and ECI	24
4.10.2.1 Input Modes and ECI Example 1	25
4.10.2.2 Input Modes and ECI Example 2	25
4.10.2.3 Input Modes and ECI Example 3	26
4.11 Batch Processing	26
4.12 Direct Output	27
4.13 Automatic Filenames	27
4.14 Working with Dots	27
4.15 Multiple Segments	28
4.16 Structured Append	28
4.17 Help Options	28
4.18 Other Output Options	29
<b>5. Using the API</b>	<b>30</b>
5.1 Creating and Deleting Symbols	30
5.2 Encoding and Saving to File	30
5.3 Encoding and Printing Functions in Depth	31
5.4 Buffering Symbols in Memory (raster)	31
5.5 Buffering Symbols in Memory (vector)	32
5.6 Setting Options	33
5.7 Handling Errors	35
5.8 Specifying a Symbology	36
5.9 Adjusting Other Output Options	36
5.10 Setting the Input Mode	37

5.11 Multiple Segments . . . . .	37
5.12 Verifying Symbology Availability . . . . .	38
5.13 Checking Symbology Capabilities . . . . .	39
5.14 Zint Version . . . . .	39
<b>6. Types of Symbology</b> . . . . .	<b>40</b>
6.1 One-Dimensional Symbols . . . . .	40
6.1.1 Code 11 . . . . .	40
6.1.2 Code 2 of 5 . . . . .	40
6.1.2.1 Standard Code 2 of 5 . . . . .	40
6.1.2.2 IATA Code 2 of 5 . . . . .	40
6.1.2.3 Industrial Code 2 of 5 . . . . .	41
6.1.2.4 Interleaved Code 2 of 5 (ISO 16390) . . . . .	41
6.1.2.5 Code 2 of 5 Data Logic . . . . .	41
6.1.2.6 ITF-14 . . . . .	41
6.1.2.7 Deutsche Post Leitcode . . . . .	42
6.1.2.8 Deutsche Post Identcode . . . . .	42
6.1.3 Universal Product Code (ISO 15420) . . . . .	42
6.1.3.1 UPC Version A . . . . .	42
6.1.3.2 UPC Version E . . . . .	43
6.1.4 European Article Number (ISO 15420) . . . . .	44
6.1.4.1 EAN-2, EAN-5, EAN-8 and EAN-13 . . . . .	44
6.1.4.2 SBN, ISBN and ISBN-13 . . . . .	45
6.1.5 Plessey . . . . .	45
6.1.5.1 UK Plessey . . . . .	45
6.1.5.2 MSI Plessey . . . . .	45
6.1.6 Telepen . . . . .	46
6.1.6.1 Telepen Alpha . . . . .	46
6.1.6.2 Telepen Numeric . . . . .	46
6.1.7 Code 39 . . . . .	46
6.1.7.1 Standard Code 39 (ISO 16388) . . . . .	46
6.1.7.2 Extended Code 39 . . . . .	46
6.1.7.3 Code 93 . . . . .	47
6.1.7.4 PZN (Pharmazentralnummer) . . . . .	47
6.1.7.5 LOGMARS . . . . .	47
6.1.7.6 Code 32 . . . . .	48
6.1.7.7 HIBC Code 39 . . . . .	48
6.1.7.8 Vehicle Identification Number (VIN) . . . . .	48
6.1.8 Codabar (EN 798) . . . . .	48
6.1.9 Pharmacode . . . . .	49
6.1.10 Code 128 . . . . .	49
6.1.10.1 Standard Code 128 (ISO 15417) . . . . .	49
6.1.10.2 Code 128 Subset B . . . . .	49
6.1.10.3 GS1-128 . . . . .	49
6.1.10.4 EAN-14 . . . . .	50
6.1.10.5 NVE-18 (SSCC-18) . . . . .	50
6.1.10.6 HIBC Code 128 . . . . .	50
6.1.10.7 DPD Code . . . . .	51
6.1.11 GS1 DataBar (ISO 24724) . . . . .	51
6.1.11.1 GS1 DataBar Omnidirectional and GS1 DataBar Truncated . . . . .	51
6.1.11.2 GS1 DataBar Limited . . . . .	51
6.1.11.3 GS1 DataBar Expanded . . . . .	52
6.1.12 Korea Post Barcode . . . . .	52
6.1.13 Channel Code . . . . .	52
6.2 Stacked Symbologies . . . . .	53
6.2.1 Basic Symbol Stacking . . . . .	53
6.2.2 Codablock-F . . . . .	53
6.2.3 Code 16K (EN 12323) . . . . .	54
6.2.4 PDF417 (ISO 15438) . . . . .	54
6.2.5 Compact PDF417 (ISO 15438) . . . . .	54
6.2.6 MicroPDF417 (ISO 24728) . . . . .	54

6.2.7 GS1 DataBar Stacked (ISO 24724)	55
6.2.7.1 GS1 DataBar Stacked	55
6.2.7.2 GS1 DataBar Stacked Omnidirectional	55
6.2.7.3 GS1 DataBar Expanded Stacked	55
6.2.8 Code 49	56
6.3 Composite Symbols (ISO 24723)	57
6.3.1 CC-A	58
6.3.2 CC-B	58
6.3.3 CC-C	58
6.4 Two-Track Symbols	59
6.4.1 Two-Track Pharmacode	59
6.4.2 POSTNET	59
6.4.3 PLANET	59
6.5 4-State Postal Codes	60
6.5.1 Australia Post 4-State Symbols	60
6.5.1.1 Customer Barcodes	60
6.5.1.2 Reply Paid Barcode	60
6.5.1.3 Routing Barcode	60
6.5.1.4 Redirect Barcode	60
6.5.2 Dutch Post KIX Code	60
6.5.3 Royal Mail 4-State Customer Code (RM4SCC)	61
6.5.4 Royal Mail 4-State Mailmark	61
6.5.5 USPS Intelligent Mail	61
6.5.6 Japanese Postal Code	61
6.6 Matrix Symbols	62
6.6.1 Data Matrix (ISO 16022)	62
6.6.2 QR Code (ISO 18004)	63
6.6.3 Micro QR Code (ISO 18004)	64
6.6.4 Rectangular Micro QR Code (rMQR)	65
6.6.5 UPNQR (Univerzalnega Plačilnega Naloga QR)	66
6.6.6 MaxiCode (ISO 16023)	66
6.6.7 Aztec Code (ISO 24778)	67
6.6.8 Aztec Runes (ISO 24778)	68
6.6.9 Code One	69
6.6.10 Grid Matrix	69
6.6.11 DotCode	70
6.6.12 Han Xin Code (ISO 20830)	71
6.6.13 Ultracode	72
6.7 Other Barcode-Like Markings	73
6.7.1 Facing Identification Mark (FIM)	73
6.7.2 Flattermarken	73
6.7.3 DAFT Code	73
<b>7. Legal and Version Information</b>	<b>74</b>
7.1 License	74
7.2 Patent Issues	74
7.3 Version Information	74
7.4 Sources of Information	74
7.5 Standards Compliance	74
<b>A. Character Encoding</b>	<b>76</b>
A.1 ASCII Standard	76
A.2 Latin Alphabet No. 1 (ISO/IEC 8859-1)	76
<b>B. CLI Help</b>	<b>77</b>

# 1. Introduction

The Zint project aims to provide a complete cross-platform open source barcode generating solution. The package currently consists of a Qt based GUI, a CLI command line executable and a library with an API to allow developers access to the capabilities of Zint. It is hoped that Zint provides a solution which is flexible enough for professional users while at the same time takes care of as much of the processing as possible to allow easy translation from input data to barcode image.

The library which forms the main component of the Zint project is currently able to encode data in over 50 barcode symbologies (types of barcode), for each of which it is possible to translate that data from either UTF-8 (Unicode) or a raw 8-bit data stream. The image can be rendered as either a Portable Network Graphic (PNG) image, Windows Bitmap (BMP), Graphics Interchange Format (GIF), ZSoft Paintbrush image (PCX), Tagged Image File Format (TIF), Enhanced Metafile Format (EMF), as Encapsulated PostScript (EPS), or as a Scalable Vector Graphic (SVG). Many options are available for setting the characteristics of the output image including the size and colour of the image, the amount of error correction used in the symbol and the orientation of the image.

## 1.1 Glossary

Some of the words and phrases used in this document are specific to barcoding, and so a brief explanation is given to help understanding:

**symbol:**

A symbol is an image which encodes data according to one of the standards. This encompasses barcodes (linear symbols) as well as any of the other methods of representing data used in this program.

**symbology:**

A method of encoding data to create a certain type of symbol.

**linear:**

A linear or one-dimensional symbol is one which consists of bars and spaces, and is what most people associate with the term 'barcode'. Examples include Code 128.

**stacked:**

A stacked symbol consists of multiple linear symbols placed one above another and which together hold the message, usually alongside some error correction data. Examples include PDF417.

**matrix:**

A matrix symbol is one based on a (usually square) grid of elements called modules. Examples include Data Matrix, but MaxiCode and DotCode are also considered matrix symbologies.

**composite:**

A composite symbology is one which is made up of elements which are both linear and stacked. Those currently supported are made up of a linear 'primary' message above which is printed a stacked component based on the PDF417 symbology. These symbols also have a separator which separates the linear and the stacked components.

**X-dimension:**

The X-dimension of a symbol is the size (usually the width) of the smallest element. For a linear symbology this is the width of the smallest bar. For matrix symbologies it is the width of the smallest module (usually a square). Barcode widths and heights are expressed in multiples of the X-dimension. Most linear symbologies can have their height varied whereas most matrix symbologies have a fixed width-to-height ratio where the height is determined by the width.

**GS1 data:**

This is a structured way of representing information which consists of 'chunks' of data, each of which starts with an Application Identifier (AI). The AI identifies what type of information is being encoded.

**Reader Initialisation:**

Some symbologies allow a special character to be included which can be detected by the scanning equipment as signifying that the data is used to program or change settings in that equipment. This data is usually not passed on to the software which handles normal input data. This feature should only be used if you are familiar with the programming codes relevant to your scanner.

**ECI:**

The Extended Channel Interpretations (ECI) mechanism allows for multi-language data to be encoded in symbols which would usually support only Latin-1 (ISO/IEC 8859-1 plus ASCII) characters. This can be useful, for example, if you need to encode Cyrillic characters, but should be used with caution as not all scanners support this method.

Two other concepts that are important are raster and vector.

**raster:**

A low level bitmap representation of an image. BMP, GIF, PCX, PNG and TIF are raster file formats.

**vector:**

A high level command- or data-based representation of an image. EMF, EPS and SVG are vector file formats. They require renderers to turn them into bitmaps.

## 2. Installing Zint

### 2.1 Linux

The easiest way to configure compilation is to take advantage of the CMake utilities. You will need to install CMake and `libpng-dev` first. For instance on apt systems:

```
sudo apt install git cmake build-essential libpng-dev
```

If you want to take advantage of Zint Barcode Studio you will also need to have Qt and its component "Desktop gcc 64-bit" installed, as well as mesa. For details see "README.linux" in the project root directory.

Once you have fulfilled these requirements unzip the source code tarball or clone the latest source

```
git clone https://git.code.sf.net/p/zint/code zint
```

and follow these steps in the top directory:

```
mkdir build
cd build
cmake ..
make
sudo make install
```

The CLI command line program can be accessed by typing

```
zint [options]
```

The GUI can be accessed by typing

```
zint-qt
```

To test that the installation has been successful a shell script is included in the "frontend" sub-directory. To run the test type

```
./test.sh
```

This should create numerous files in the sub-directory "frontend/test\_sh\_out" showing the many modes of operation which are available from Zint.

### 2.2 Microsoft Windows

For Microsoft Windows, Zint is distributed as a binary executable. Simply download the ZIP file, then right-click on the ZIP file and "Extract All". A new folder will be created within which are two binary files:

- `qtZint.exe` - Zint Barcode Studio
- `zint.exe` - Command Line Interface

For fresh releases you will get a warning message from Microsoft Defender SmartScreen that this is an 'unrecognised app'. This happens because Zint is a free and open-source software project with no advertising and hence no income, meaning we are not able to afford the \$664 per year to have the application digitally signed by Microsoft.

To build Zint on Windows from source, see "win32/README".

### 2.3 Apple macOS

Zint can be installed using Homebrew. To install Homebrew input the following line into the macOS terminal

```
/usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Once Homebrew is installed use the following command to install Zint.

```
brew install zint
```

### 2.4 Zint Tcl Backend

The Tcl backend in the "backend\_tcl" sub-directory may be built using the provided TEA (Tcl Extension Architecture) build on Linux, Windows, macOS and Android. For Windows, an MSVC6 makefile is also available.

### 3. Using Zint Barcode Studio

Zint Barcode Studio is the graphical user interface for Zint. If you are starting from a command line interface you can start the GUI by typing

```
zint-qt
```

or on Windows

```
qtZint.exe
```

See the note in section [2.2 Microsoft Windows](#) about Microsoft Defender SmartScreen.

Below is a brief guide to Zint Barcode Studio.

#### 3.1 Main Window and Data Tab

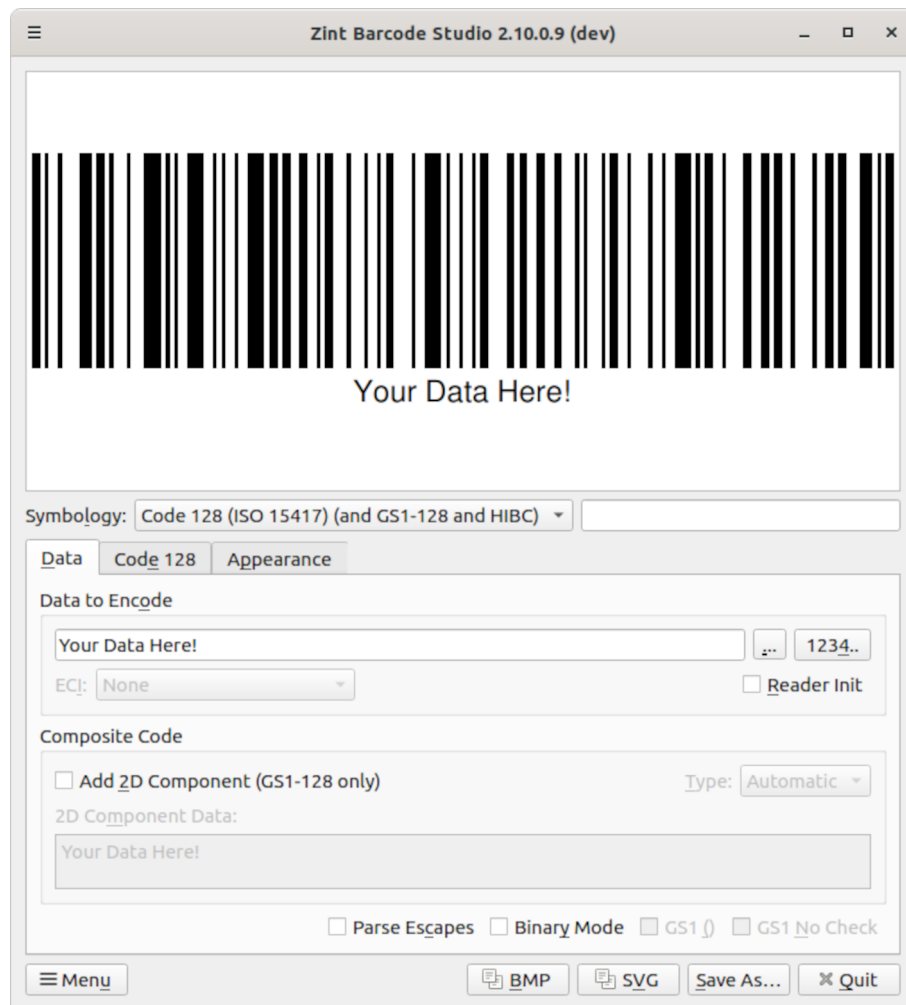


Figure 1: Zint Barcode Studio on startup - main window with Data tab

This is the main window of Zint Barcode Studio. The top of the window shows a preview of the barcode which the current settings would create. These settings can be changed using the controls below. The text box in the "Data to Encode" groupbox on this first Data tab allows you to enter the data to be encoded. When you are happy with your settings you can use the "Save As" button to save the resulting image to a file.

The "Symbolology" drop-down box gives access to all of the symbologies supported by Zint shown in alphabetical order. The text box to its right can filter the drop-down to only show matching symbologies. For instance typing "mail" will only show barcodes in the drop-down whose names contain the word "mail". Each word entered will match. So typing "mail post" will show barcodes whose names contain "mail" or "post" (or both).

The "BMP" and "SVG" buttons at the bottom will copy the image to the clipboard in BMP format and SVG format respectively. Further copy-to-clipboard formats are available by clicking the "Menu" button, along with "CLI



Equivalent", "Save As", "Help", "About" and "Quit" options. Most of the options are also available in a context menu by right-clicking the preview.

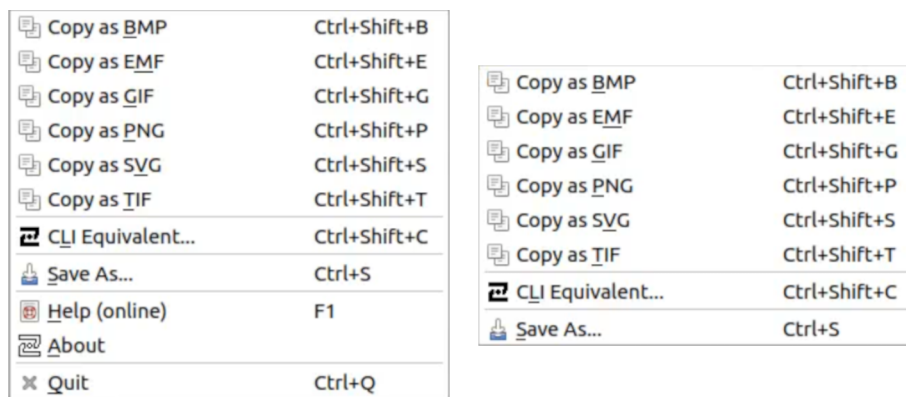


Figure 2: Zint Barcode Studio main menu (left) and context menu (right)

### 3.2 Composite Groupbox

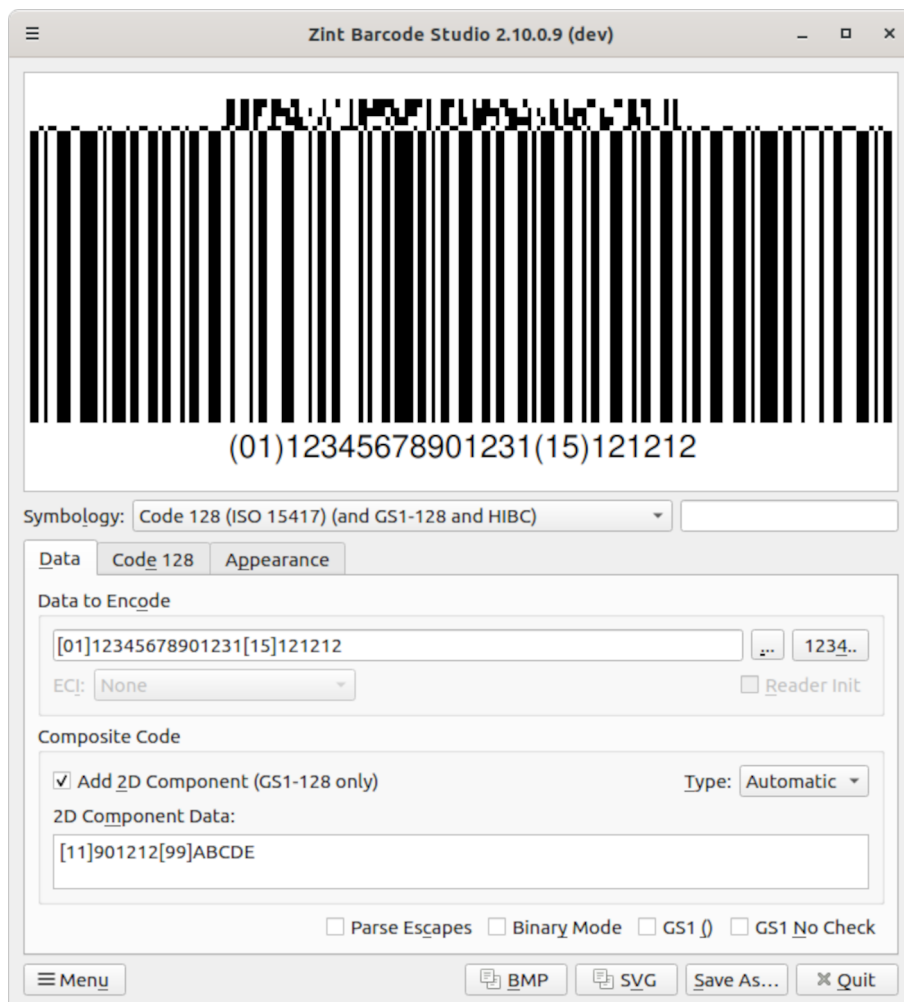


Figure 3: Zint Barcode Studio encoding GS1 composite data

In the middle of the Data tab is an area for creating composite symbologies which appears when the currently selected symbology is supported by the composite symbology standard. GS1 data can then be entered with square brackets used to separate Application Identifier (AI) information from data as shown here. For details, see [6.3 Composite Symbols \(ISO 24723\)](#).

### 3.3 Additional ECI/Data Segments Groupbox



Figure 4: Zint Barcode Studio encoding multiple segments

For symbologies that support ECIs (Extended Channel Interpretations) the middle of the Data tab is an area for entering additional data segments with their own ECIs. Up to 4 segments (including the main "Data to Encode" as segment 0) may be specified. See [4.15 Multiple Segments](#) for details.

### 3.4 Symbology-specific Tab



Figure 5: Zint Barcode Studio showing Aztec Code options

For a number of symbolologies extra options are available to fine-tune the format, appearance and content of the symbol generated. These are given in a second tab.

Here the method is shown for adjusting the size or error correction level of an Aztec Code symbol, selecting how its data is to be treated, and setting it as part of a Structured Append sequence of symbols.

### 3.5 Appearance Tab

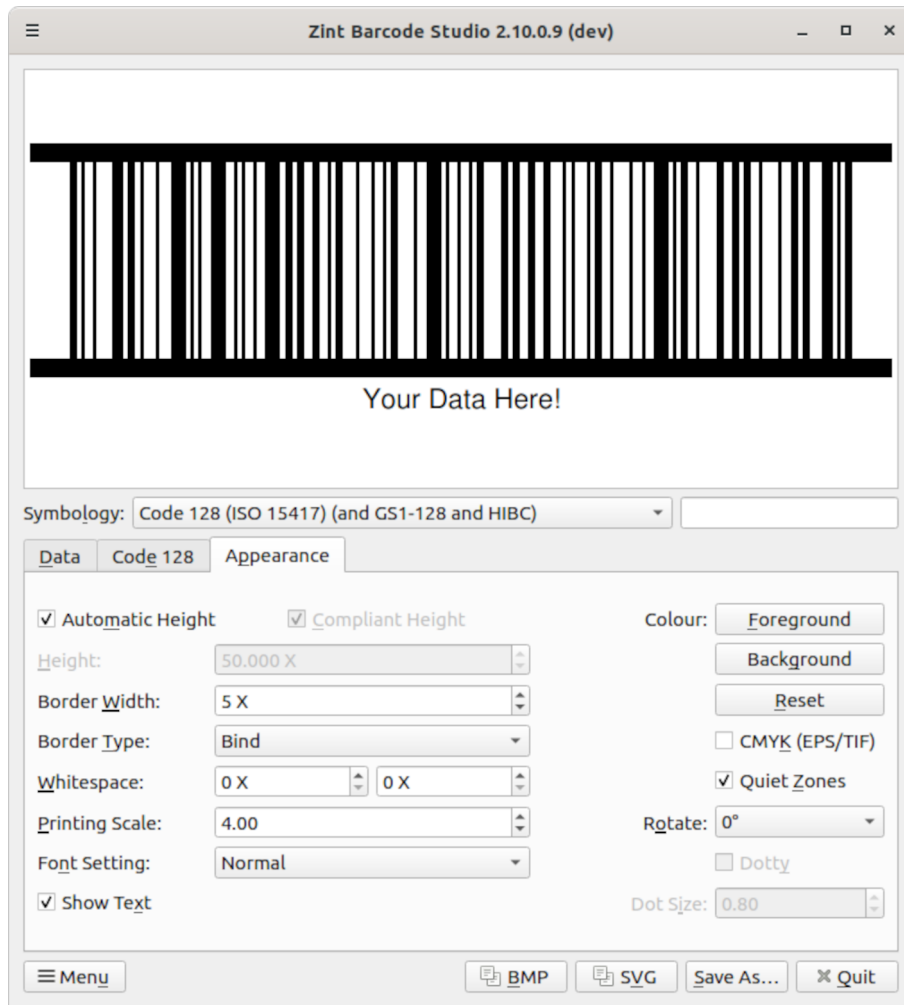


Figure 6: Zint Barcode Studio showing Appearance tab options

The Appearance tab can be used to adjust the dimensions and other properties of the symbol. The "Height" value affects the height of symbologies which do not have a fixed width-to-height ratio, i.e. those other than matrix symbologies. Boundary bars ("Border Type") can be added and adjusted and the size of the saved image ("Printing Scale") can be determined.

### 3.6 Colour Dialog

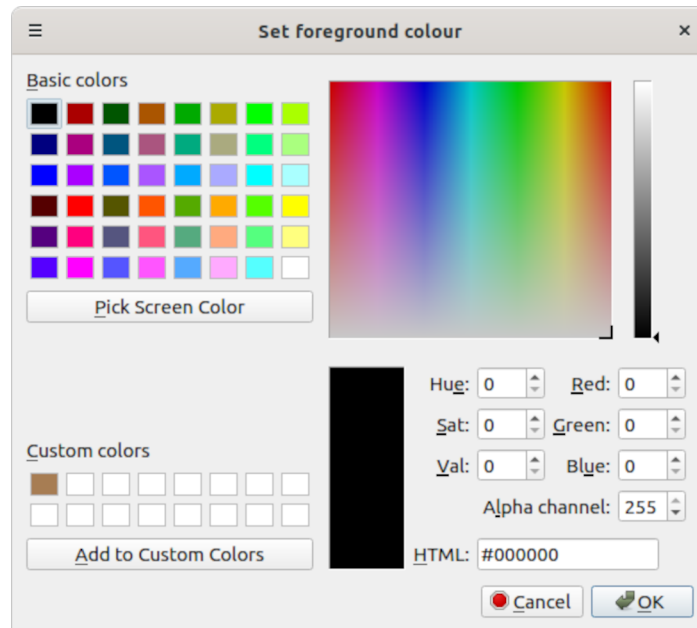


Figure 7: The colour picker tool

A colour dialog is used to adjust the colour of the foreground and background of the generated image. In the Appearance tab click on the "Foreground" or "Background" button respectively. The colours can be reset to black-on-white using the "Reset" button.

### 3.7 Data Dialog

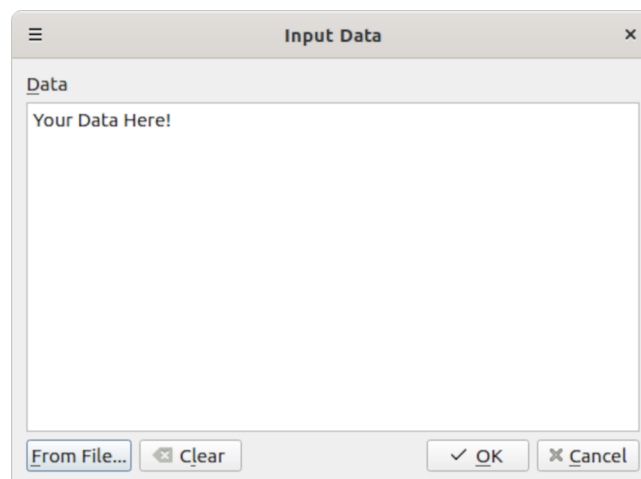


Figure 8: Entering longer text input

Clicking on the ellipsis "... " button next to the "Data to Encode" text box in the Data tab opens a larger window which can be used to enter longer strings of text. You can also use this window to load data from a file.

The dialog is also available for additional ECI/Data segments by clicking the ellipsis button to the right of their data text boxes.

Note that if your data contains line feeds (LF) then the data will be split into separate lines in the dialog box. On saving the data back to the main text box any separate lines in the data will be escaped as '\n' and the "Parse Escapes" checkbox will be set. This only affects line feeds, not carriage returns (CR) or CR+LF pairs, and behaves the same on both Windows and Unix. (For details on escape sequences, see [4.1 Inputting Data.](#))

### 3.8 Sequence Dialog

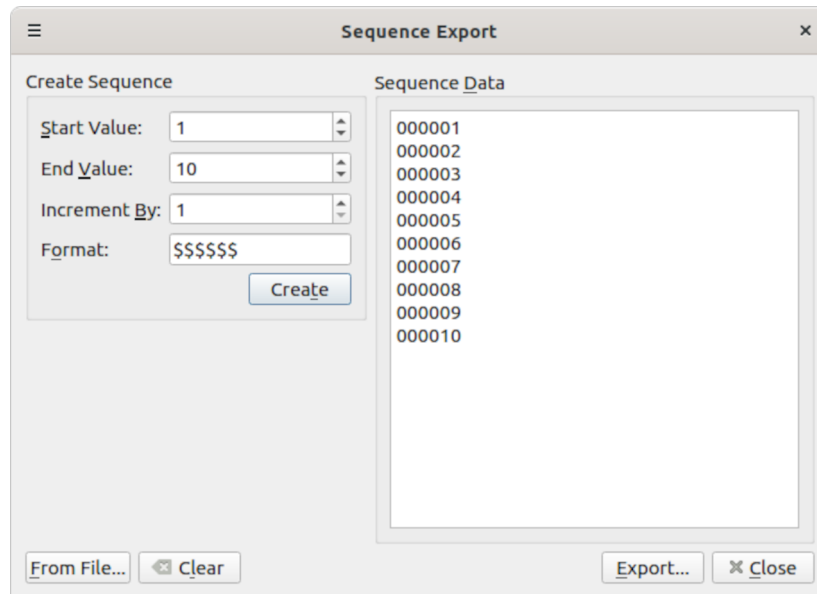


Figure 9: Creating a sequence of barcode symbols

Clicking on the sequence button (labelled "1234 . .") in the Data tab opens the Sequence Dialog. This allows you to create multiple barcode images by entering a sequence of data inputs in the right hand panel. Sequences can also be automatically generated by entering parameters on the left hand side or by importing the data from a file. Zint will generate a separate barcode image for each line of text in the right hand panel. The format field determines the format of the automatically generated sequence where characters have the meanings as given below:

Table : Sequence Format Characters

Character	Effect
#	Insert leading spaces
\$	Insert leading zeroes
*	Insert leading asterisks
Any other character	Interpreted literally

### 3.9 Export Dialog

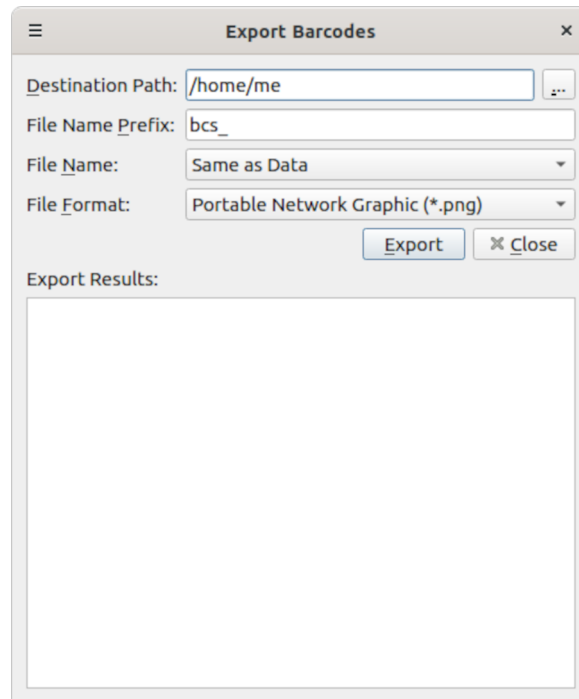


Figure 10: Setting filenames for an exported sequence of barcode symbols

The Export Dialog invoked by pressing the "Export" button in the Sequence Dialog sets the parameters for exporting a sequence of barcode images. Here you can set the filename and the output image format. Note that the symbology, colour and other formatting information are taken from the main window.

### 3.10 CLI Equivalent Dialog

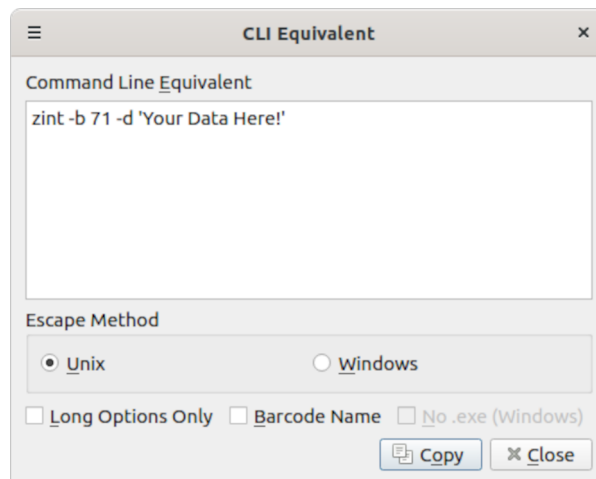


Figure 11: CLI Equivalent Dialog

The "CLI Equivalent" dialog can be invoked from the main menu or the context menu and displays the CLI command that will reproduce the barcode as currently configured in the GUI. Press the "Copy" button to copy the command to the clipboard, which can then be pasted into the command line.

## 4. Using the Command Line

This section describes how to encode data using the command line frontend program. The examples given are for the Unix platform, but the same options are available for Windows - just remember to include the executable file extension if ".EXE" is not in your PATHEXT environment variable, i.e.:

```
zint.exe -d "This Text"
```

For compatibility with Windows the examples use double quotes to delimit data, though on Unix single quotes are generally preferable as they stop the shell from processing any characters such as backslash or dollar. A single quote itself is dealt with by terminating the single-quoted text, backslashing the single quote, and then continuing:

```
zint -d 'Text containing a single quote \' in the middle'
```

Some examples use backslash (\) to continue commands onto the next line. For Windows, use caret (^) instead.

Certain options that take values have short names as well as long ones, namely -b (--barcode), -d (--data), -i (--input), -o (--output) and -w (--whitesp). For these a space should be used to separate the short name from its value, to avoid ambiguity. For long names a space or an equals sign may be used. For instance:

```
zint -d "This Text"
zint --data="This Text"
zint --data "This Text"
```

The examples use a space separator for short option names, and an equals sign for long option names.

### 4.1 Inputting Data

The data to encode can be entered at the command line using the -d or --data option, for example

```
zint -d "This Text"
```

This will encode the text "This Text". Zint will use the default symbology, Code 128, and output to the default file "out.png" in the current directory. Alternatively, if libpng was not present when Zint was built, the default output file will be "out.gif".

The data input to the Zint CLI is assumed to be encoded in UTF-8 (Unicode) format (Zint will correctly handle UTF-8 data on Windows). If you are encoding characters beyond the 7-bit ASCII set using a scheme other than UTF-8 then you will need to set the appropriate input options as shown in [4.10 Input Modes](#) below.

Non-printing characters can be entered on the command line using backslash (\) as an escape character in combination with the --esc switch. Permissible sequences are shown in the table below.

Table : Escape Sequences

Escape Sequence	ASCII Equivalent	Name	Interpretation
\0	0x00	NUL	Null character
\E	0x04	EOT	End of Transmission
\a	0x07	BEL	Bell
\b	0x08	BS	Backspace
\t	0x09	HT	Horizontal Tab
\n	0x0A	LF	Line Feed
\v	0x0B	VT	Vertical Tab
\f	0x0C	FF	Form Feed
\r	0x0D	CR	Carriage Return
\e	0x1B	ESC	Escape
\G	0x1D	GS	Group Separator
\R	0x1E	RS	Record Separator
\\	0x5C	\	Backslash
\xNN	0xNN		Any 8-bit character where NN is hexadecimal
\uNNNN			Any 16-bit Unicode BMP <sup>1</sup> character where NNNN is hexadecimal

<sup>1</sup>In Unicode contexts, BMP stands for Basic Multilingual Plane, the plane 0 codeset from U+0000 to U+D7FF and U+E000 to U+FFFF (i.e. excluding surrogates). Not to be confused with the Windows Bitmap file format BMP!



Input data can be read directly from file using the `-i` or `--input` switch as shown below. The input file is assumed to be UTF-8 formatted unless an alternative mode is selected. This command replaces the use of the `-d` switch.

```
zint -i somefile.txt
```

Note that except when batch processing (see [4.11 Batch Processing](#) below), the file should not end with a newline (LF on Unix, CR+LF on Windows) unless you want the newline to be encoded in the symbol.

## 4.2 Directing Output

Output can be directed to a file other than the default using the `-o` or `--output` switch. For example:

```
zint -o here.png -d "This Text"
```

This draws a Code 128 barcode in the file `"here.png"`. If an Encapsulated PostScript file is needed simply append the filename with `".eps"`, and so on for the other supported file types:

```
zint -o there.eps -d "This Text"
```

## 4.3 Selecting Barcode Type

Selecting which type of barcode you wish to produce (i.e. which symbology to use) can be done at the command line using the `-b` or `--barcode` switch followed by the appropriate integer value or name in the following table. For example to create a Data Matrix symbol you could use:

```
zint -b 71 -o datamatrix.png -d "Data to encode"
```

or

```
zint -b DATAMATRIX -o datamatrix.png -d "Data to encode"
```

Names are treated case-insensitively by the CLI, and the `BARCODE_` prefix and any underscores are optional.

Table : Barcode Types (Symbologies)

Numeric Value	Name <sup>2</sup>	Barcode Name
1	BARCODE_CODE11	Code 11
2*	BARCODE_C25STANDARD	Standard Code 2 of 5
3	BARCODE_C25INTER	Interleaved 2 of 5
4	BARCODE_C25IATA	Code 2 of 5 IATA
6	BARCODE_C25LOGIC	Code 2 of 5 Data Logic
7	BARCODE_C25IND	Code 2 of 5 Industrial
8	BARCODE_CODE39	Code 3 of 9 (Code 39)
9	BARCODE_EXCODE39	Extended Code 3 of 9 (Code 39+)
13	BARCODE_EANX	EAN (including EAN-8 and EAN-13)
14	BARCODE_EANX_CHK	EAN + Check Digit
16*	BARCODE_GS1_128	GS1-128 (UCC.EAN-128)
18	BARCODE_CODABAR	Codabar
20	BARCODE_CODE128	Code 128 (automatic subset switching)
21	BARCODE_DPLEIT	Deutsche Post Leitcode
22	BARCODE_DPIDENT	Deutsche Post Identcode
23	BARCODE_CODE16K	Code 16K
24	BARCODE_CODE49	Code 49
25	BARCODE_CODE93	Code 93
28	BARCODE_FLAT	Flattermarken
29*	BARCODE_DBAR_OMN	GS1 DataBar Omnidirectional (including GS1 DataBar Truncated)
30*	BARCODE_DBAR_LTD	GS1 DataBar Limited
31*	BARCODE_DBAR_EXP	GS1 DataBar Expanded
32	BARCODE_TELEPEN	Telepen Alpha
34	BARCODE_UPCA	UPC-A
35	BARCODE_UPCA_CHK	UPC-A + Check Digit

Numeric Value	Name	Barcode Name
37	BARCODE_UPCE	UPC-E
38	BARCODE_UPCE_CHK	UPC-E + Check Digit
40	BARCODE_POSTNET	POSTNET
47	BARCODE_MSI_PLESSEY	MSI Plessey
49	BARCODE_FIM	FIM
50	BARCODE_LOGMARS	LOGMARS
51	BARCODE_PHARMA	Pharmacode One-Track
52	BARCODE_PZN	PZN
53	BARCODE_PHARMA_TWO	Pharmacode Two-Track
55	BARCODE_PDF417	PDF417
56*	BARCODE_PDF417COMP	Compact PDF417 (Truncated PDF417)
57	BARCODE_MAXICODE	MaxiCode
58	BARCODE_QRCODE	QR Code
60	BARCODE_CODE128B	Code 128 (Subset B)
63	BARCODE_AUSPOST	Australia Post Standard Customer
66	BARCODE_AUSREPLY	Australia Post Reply Paid
67	BARCODE_AUSROUTE	Australia Post Routing
68	BARCODE_AUSDIRECT	Australia Post Redirection
69	BARCODE_ISBNX	ISBN (EAN-13 with verification stage)
70	BARCODE_RM4SCC	Royal Mail 4-State (RM4SCC)
71	BARCODE_DATAMATRIX	Data Matrix (ECC200)
72	BARCODE_EAN14	EAN-14
73	BARCODE_VIN	Vehicle Identification Number
74	BARCODE_CODABLOCKF	Codablock-F
75	BARCODE_NVE18	NVE-18 (SSCC-18)
76	BARCODE_JAPANPOST	Japanese Postal Code
77	BARCODE_KOREAPOST	Korea Post
79*	BARCODE_DBAR_STK	GS1 DataBar Stacked
80*	BARCODE_DBAR_OMNSTK	GS1 DataBar Stacked Omnidirectional
81*	BARCODE_DBAR_EXPSTK	GS1 DataBar Expanded Stacked
82	BARCODE_PLANET	PLANET
84	BARCODE_MICROPDF417	MicroPDF417
85*	BARCODE_USPS_IMAIL	USPS Intelligent Mail (OneCode)
86	BARCODE_PLESSEY	UK Plessey
87	BARCODE_TELEPEN_NUM	Telepen Numeric
89	BARCODE_ITF14	ITF-14
90	BARCODE_KIX	Dutch Post KIX Code
92	BARCODE_AZTEC	Aztec Code
93	BARCODE_DAFT	DAFT Code
96	BARCODE_DPD	DPD Code
97	BARCODE_MICROQR	Micro QR Code
98	BARCODE_HIBC_128	HIBC Code 128
99	BARCODE_HIBC_39	HIBC Code 39
102	BARCODE_HIBC_DM	HIBC Data Matrix ECC200
104	BARCODE_HIBC_QR	HIBC QR Code
106	BARCODE_HIBC_PDF	HIBC PDF417
108	BARCODE_HIBC_MICPDF	HIBC MicroPDF417
110	BARCODE_HIBC_BLOCKF	HIBC Codablock-F
112	BARCODE_HIBC_AZTEC	HIBC Aztec Code
115	BARCODE_DOTCODE	DotCode
116	BARCODE_HANXIN	Han Xin (Chinese Sensible) Code
121	BARCODE_MAILMARK	Royal Mail 4-state Mailmark
128	BARCODE_AZRUNE	Aztec Runes
129	BARCODE_CODE32	Code 32
130	BARCODE_EANX_CC	Composite Symbol with EAN linear component
131*	BARCODE_GS1_128_CC	Composite Symbol with GS1-128 linear component
132*	BARCODE_DBAR_OMN_CC	Composite Symbol with GS1 DataBar Omnidirectional linear component

Numeric Value	Name	Barcode Name
133*	BARCODE_DBAR_LTD_CC	Composite Symbol with GS1 DataBar Limited linear component
134*	BARCODE_DBAR_EXP_CC	Composite Symbol with GS1 DataBar Expanded linear component
135	BARCODE_UPCA_CC	Composite Symbol with UPC-A linear component
136	BARCODE_UPCE_CC	Composite Symbol with UPC-E linear component
137*	BARCODE_DBAR_STK_CC	Composite Symbol with GS1 DataBar Stacked component
138*	BARCODE_DBAR_OMNSTK_CC	Composite Symbol with GS1 DataBar Stacked Omnidirectional component
139*	BARCODE_DBAR_EXPSTK_CC	Composite Symbol with GS1 DataBar Expanded Stacked component
140	BARCODE_CHANNEL	Channel Code
141	BARCODE_CODEONE	Code One
142	BARCODE_GRIDMATRIX	Grid Matrix
143	BARCODE_UPNQR	UPNQR (Univerzalnega Plačilnega Naloga QR)
144	BARCODE_ULTRA	Ultracode
145	BARCODE_RMQR	Rectangular Micro QR Code (rMQR)

## 4.4 Adjusting Height

The height of a symbol (except those with a fixed width-to-height ratio) can be adjusted using the `--height` switch. For example:

```
zint --height=100 -d "This Text"
```

This specifies a symbol height of 100 times the X-dimension of the symbol.

The default height of most linear barcodes is 50X, but this can be changed for barcodes whose specifications give a standard height by using the switch `--compliantheight`. For instance

```
zint -b LOGMARS -d "This Text" --compliantheight
```

will produce a barcode of height 45.455X instead of the normal default of 50X. The flag also causes Zint to return a warning if a non-compliant height is given:

```
zint -b LOGMARS -d "This Text" --compliantheight --height=6.2
```

Warning 247: Height not compliant with standards

Another switch is `--heightperrow`, which can be useful for symbologies that have a variable number of linear rows, namely Codablock-F, Code 16K, Code 49, GS1 DataBar Expanded Stacked, MicroPDF417 and PDF417, as it changes the treatment of the height value from overall height to per-row height, allowing you to specify a consistent height for each linear row without having to know how many there are. For instance

```
zint -b PDF417 -d "This Text" --height=4 --heightperrow
```



Figure 12: `zint -b PDF417 -d "This Text" --height=4 --heightperrow`

will produce a barcode of height 32X, with each of the 8 rows 4X high.

## 4.5 Adjusting Whitespace

The amount of horizontal whitespace to the left and right of the generated barcode can be altered using the `-w` or `--whitespace` switch. For example:

<sup>2</sup>The symbologies marked with an asterisk (\*) in the above table used different names in Zint before version 2.9.0. For example, symbology 29 used the name `BARCODE_RSS14`. These names are now deprecated but are still recognised by Zint and will continue to be supported in future versions.

```
zint -w 10 -d "This Text"
```

This specifies a whitespace width of 10 times the X-dimension of the symbol both to the left and to the right of the barcode.

The amount of vertical whitespace above and below the barcode can be altered using the `--vwhitespace` switch. For example for 3 times the X-dimension:

```
zint --vwhitespace=3 -d "This Text"
```

Note that the whitespace at the bottom appears below the text, if any.

Horizontal and vertical whitespace can of course be used together:

```
zint -b DATAMATRIX --whitespace=1 --vwhitespace=1 -d "This Text"
```

A `--quietzones` option is also available which adds quiet zones compliant with the symbology's specification. This is in addition to any whitespace specified with the `--whitespace` or `--vwhitespace` switches.

Note that Codablock-F, Code 16K, Code 49, ITF-14, EAN-13, EAN-8, EAN-5, EAN-2, ISBN, UPC-A and UPC-E have compliant quiet zones added by default. This can be disabled with the option `--noquietzones`.

## 4.6 Adding Boundary Bars and Boxes

Zint allows the symbol to be bound with 'boundary bars' (also known as 'bearer bars') using the option `--bind`. These bars help to prevent misreading of the symbol by corrupting a scan if the scanning beam strays off the top or bottom of the symbol. Zint can also put a border right around the symbol and its horizontal whitespace with the `--box` option.

The width of the boundary or box must be specified using the `--border` switch. For example:

```
zint --box --border=10 -w 10 -d "This Text"
```

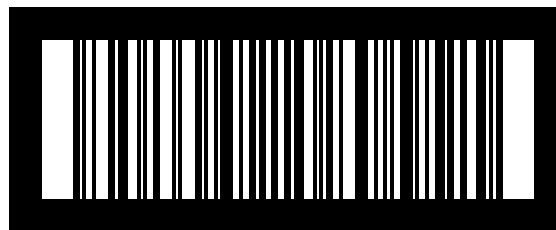


Figure 13: `zint --border=10 --box -d "This Text" -w 10`

gives a box with a width 10 times the X-dimension of the symbol. Note that when specifying a box, horizontal whitespace is usually required in order to create a quiet zone between the barcode and the sides of the box.

For linear symbols, horizontal boundary bars appear tight against the barcode, inside any vertical whitespace (or text). For matrix symbols, however, where they are decorative rather than functional, boundary bars appear outside any whitespace.

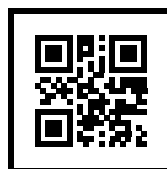


Figure 14: `zint -b QRCODE --border=1 --box -d "This Text" --quietzones`

Codablock-F, Code 16K and Code 49 always have boundary bars, and default to particular horizontal whitespace values. Special considerations apply to ITF-14 - see [6.1.2.6 ITF-14](#) for that symbology.

## 4.7 Using Colour

The default colours of a symbol are a black symbol on a white background. Zint allows you to change this. The `-r` or `--reverse` switch allows the default colours to be inverted so that a white symbol is shown on a black background (known as reflectance reversal). For example the command

```
zint -r -d "This Text"
```

gives an inverted Code 128 symbol. This is not practical for most symbologies but white-on-black is allowed by the Aztec Code, Data Matrix, Han Xin Code, Grid Matrix and QR Code symbology specifications.

For more specific needs the foreground (ink) and background (paper) colours can be specified using the `--fg` and `--bg` options followed by a number in RRGGBB hexadecimal notation (the same system used in HTML). For example the command

```
zint --fg=00FF00 -d "This Text"
```

alters the symbol to a bright green.



Figure 15: `zint -d "This Text" --fg=00FF00`

Zint also supports RGBA colour information for some output file formats which support alpha channels (currently only PNG, SVG and TIF) in a RRGGBBAA format. For example:

```
zint --fg=00ff0055 -d "This Text"
```

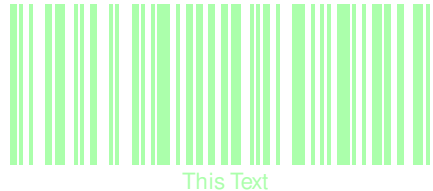


Figure 16: `zint -d "This Text" --fg=00FF0055`

will produce a semi-transparent green foreground with standard (white) background. Note that transparency is handled differently for raster and vector files so that...

```
zint --bg=ff0000 --fg=ffffff00 ...
```

will give different results for PNG and SVG. Experimentation is advised!

In addition the `--nobackground` option will simply remove the background from EMF, EPS, GIF, PNG, SVG and TIF files.

## 4.8 Rotating the Symbol

The symbol can be rotated through four orientations using the `--rotate` option followed by the angle of rotation as shown below.

```
--rotate=0 (default)
--rotate=90
--rotate=180
--rotate=270
```

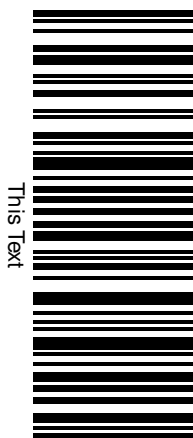


Figure 17: `zint -d "This Text" --rotate=90`

## 4.9 Adjusting Image Size

The scale of the image can be altered using the `--scale` option followed by a multiple of the default X-dimension. The scale is multiplied by 2 before being applied. The default scale is 1.

For raster output, the default X-dimension is 2 pixels (except for MaxiCode, see [4.9.2 MaxiCode Raster Scaling](#) below). For example for PNG images a scale of 5 will increase the X-dimension to 10 pixels. Scales for raster output should be given in increments of 0.5, i.e. 0.5, 1, 1.5, 2, 2.5, 3, 3.5, etc., to avoid the X-dimension varying across the symbol due to interpolation. 0.5 increments are also faster to render.

The minimum scale for non-dotty raster output is 0.5, giving a minimum X-dimension of 1 pixel, and text will not be printed for scales less than 1. The minimum scale for raster output in dotty mode is 1 (see [4.14 Working with Dots](#)).

The minimum scale for vector output is 0.1, giving a minimum X-dimension of 0.2.

The maximum scale for both raster and vector is 100.

### 4.9.1 Scaling Example

The GS1 General Specifications Section 5.2.6.6 'Symbol dimensions at nominal size' gives an example of an EAN-13 barcode using the X-dimension of 0.33mm. To print that example as a PNG at 12 dots per mm (dpmm), the equivalent of 300 dots per inch (dpi = dpmm \* 25.4), specify a scale of 2, since  $0.33 * 12 = 3.96$  pixels, or 4 pixels rounding to the nearest pixel:

```
zint -b EANX -d "501234567890" --complianceheight --scale=2
```

This will result in output of 38.27mm x 26.08mm (WxH) at 300 dpi. The following table shows the scale to use (in 0.5 increments) depending on the dpmm desired, for a target X-dimension of 0.33mm:

Table : Scaling for X-dimension 0.33mm

dpmm	dpi	scale
6	150	1
8	200	1.5
12	300	2
16	400	3
24	600	4
47	1200	8
95	2400	15.5
189	4800	31

### 4.9.2 MaxiCode Raster Scaling

For MaxiCode symbols, which use hexagons, the scale for raster output is multiplied by 10 before being applied. The minimum scale is 0.2, so the minimum X-dimension is 2 pixels.

MaxiCode symbols have fixed size ranges of 24.82mm to 27.93mm in width, and 23.71mm to 26.69mm in height, excluding quiet zones. The following table shows the scale to use depending on the dpmm desired, with dpi equivalents:

Table : MaxiCode Raster Scaling

dpmm	dpi	scale
6	150	0.5
8	200	0.7
12	300	1
16	400	1.4
24	600	2.1
47	1200	4.1
95	2400	8.2
189	4800	16.4

Note that the 0.5 increment recommended for normal raster output does not apply. Scales below 0.5 are not recommended and may produce symbols that are not within the minimum/maximum size ranges.

## 4.10 Input Modes

### 4.10.1 Unicode, Data, and GS1 Modes

By default all CLI input data is assumed to be encoded in UTF-8 format. Many barcode symbologies encode data using the Latin-1 (ISO/IEC 8859-1 plus ASCII) character set, so input is converted from UTF-8 to Latin-1 before being put in the symbol. In addition QR Code and its variants and Han Xin Code can by default encode Japanese (Kanji) or Chinese (Hanzi) characters which are also converted from UTF-8.

There are two exceptions to the Latin-1 default: Grid Matrix, whose default character set is GB 2312 (Chinese); and UPNQR, whose default character set is Latin-2 (ISO/IEC 8859-2 plus ASCII).

Table : Default Character Sets

Symbology	Default character sets	Alternate if input not Latin-1
Aztec Code	Latin-1	None
Codablock-F	Latin-1	None
Code 128	Latin-1	None
Code 16k	Latin-1	None
Code One	Latin-1	None
Data Matrix	Latin-1	None
DotCode	Latin-1	None
Grid Matrix	GB 2312 (includes ASCII)	N/A
Han Xin	Latin-1	GB 18030 (includes ASCII)
MaxiCode	Latin-1	None
MicroPDF417	Latin-1	None
Micro QR Code	Latin-1	Shift JIS (includes ASCII <sup>3</sup> )
PDF417	Latin-1	None
QR Code	Latin-1	Shift JIS (see above)
rMQR	Latin-1	Shift JIS (see above)
Ultracode	Latin-1	None
UPNQR	Latin-2	N/A
All others	ASCII	N/A

If Zint encounters characters which can not be encoded using the default character encoding then it will take advantage of the ECI (Extended Channel Interpretations) mechanism to encode the data if the symbology supports it - see [4.10.2 Input Modes and ECI](#) below.

GS1 data can be encoded in a number of symbologies. Application Identifiers (AIs) should be enclosed in

<sup>3</sup>Shift JIS (JIS X 0201 Roman) re-maps two ASCII characters: backslash (\) to the yen sign (¥), and tilde (~) to overline (U+203E).



[square brackets] followed by the data to be encoded (see 6.1.10.3 GS1-128). To encode GS1 data use the --gs1 option. GS1 mode is assumed (and doesn't need to be set) for GS1-128, EAN-14, GS1 DataBar and Composite symbologies but is also available for Aztec Code, Code 16K, Code 49, Code One, Data Matrix, DotCode, QR Code and Ultracode.

Health Industry Barcode (HIBC) data may also be encoded in the symbologies Aztec Code, Codablock-F, Code 128, Code 39, Data Matrix, MicroPDF417, PDF417 and QR Code. Within this mode, the leading '+' and the check character are automatically added, conforming to HIBC Labeler Identification Code (HIBC LIC). For HIBC Provider Applications Standard (HIBC PAS), preface the data with a slash '/ '.

The --binary option encodes the input data as given. Automatic code page translation to an ECI page is disabled, and no validation of the data's encoding takes place. This may be used for raw binary or binary encrypted data. This switch plays together with the built-in ECI logic and examples may be found below.

The --fullmultibyte option uses the multibyte modes of QR Code, Micro QR Code, Rectangular Micro QR Code, Han Xin Code and Grid Matrix for non-ASCII data, maximizing density. This is achieved by using compression designed for Kanji/Hanzi characters; however some decoders take blocks which are encoded this way and interpret them as Kanji/Hanzi characters, thus causing data corruption. Symbols encoded with this option should be checked against decoders before they are used. The popular open-source ZXing decoder is known to exhibit this behaviour.

#### 4.10.2 Input Modes and ECI

If your data contains characters that are not in the default character set, you may encode it using an ECI-aware symbology and an ECI value from Table : ECI Codes below. The ECI information is added to your code symbol as prefix data. The symbologies that support ECI are

Table : ECI-Aware Symbologies

Aztec Code	DotCode	MaxiCode	QR Code
Code One	Grid Matrix	MicroPDF417	rMQR
Data Matrix	Han Xin Code	PDF417	Ultracode

Be aware that not all barcode readers support ECI mode, so this can sometimes lead to unreadable barcodes. If you are using characters beyond those supported by the default character set then you should check that the resulting barcode can be understood by your target barcode reader.

The ECI value may be specified with the --eci switch, followed by the value in the column "ECI Code". The input data should be UTF-8 formatted. Zint automatically translates the data into the target encoding.

Table : ECI Codes

ECI Code	Character Encoding Scheme (ISO/IEC 8859 schemes include ASCII)
3	ISO/IEC 8859-1 - Latin alphabet No. 1
4	ISO/IEC 8859-2 - Latin alphabet No. 2
5	ISO/IEC 8859-3 - Latin alphabet No. 3
6	ISO/IEC 8859-4 - Latin alphabet No. 4
7	ISO/IEC 8859-5 - Latin/Cyrillic alphabet
8	ISO/IEC 8859-6 - Latin/Arabic alphabet
9	ISO/IEC 8859-7 - Latin/Greek alphabet
10	ISO/IEC 8859-8 - Latin/Hebrew alphabet
11	ISO/IEC 8859-9 - Latin alphabet No. 5 (Turkish)
12	ISO/IEC 8859-10 - Latin alphabet No. 6 (Nordic)
13	ISO/IEC 8859-11 - Latin/Thai alphabet
15	ISO/IEC 8859-13 - Latin alphabet No. 7 (Baltic)
16	ISO/IEC 8859-14 - Latin alphabet No. 8 (Celtic)
17	ISO/IEC 8859-15 - Latin alphabet No. 9
18	ISO/IEC 8859-16 - Latin alphabet No. 10
20	Shift JIS (JIS X 0208 and JIS X 0201)
21	Windows 1250 - Latin 2 (Central Europe)
22	Windows 1251 - Cyrillic



ECI Code	Character Encoding Scheme (ISO/IEC 8859 schemes include ASCII)
23	Windows 1252 - Latin 1
24	Windows 1256 - Arabic
25	UTF-16BE (High order byte first)
26	UTF-8
27	ASCII (ISO/IEC 646 IRV)
28	Big5 (Taiwan) Chinese Character Set
29	GB 2312 (PRC) Chinese Character Set
30	Korean Character Set EUC-KR (KS X 1001:2002)
31	GBK Chinese Character Set
32	GB 18030 Chinese Character Set
33	UTF-16LE (Low order byte first)
34	UTF-32BE (High order bytes first)
35	UTF-32LE (Low order bytes first)
170	ISO/IEC 646 Invariant <sup>4</sup>
899	8-bit binary data

An ECI value of 0 does not encode any ECI information in the code symbol (unless the data contains non-default character set characters). In this case, the default character set applies (see Table : **Default Character Sets** above).

If no ECI is specified or a value of 0 is given, and the data does contain characters other than in the default character set, then Zint will automatically insert the appropriate single-byte ECI if possible (ECIs 3 to 24, excluding ECI 20), or failing that ECI 26 (UTF-8). A warning will be generated. This mechanism is not applied if the `--binary` option is given.

Multiple ECIs can be specified using the `--segN` options - see **4.15 Multiple Segments**.

Note: the `--eci=3` specification should only be used for special purposes. Using this parameter, the ECI information is explicitly added to the symbol. Nevertheless, for ECI Code 3, this is not usually required, as this is the default encoding for most barcodes, which is also active without any ECI information.

#### 4.10.2.1 Input Modes and ECI Example 1

The Euro sign U+20AC can be encoded in ISO/IEC 8859-15. The Euro sign has the ISO/IEC 8859-15 codepoint hex "A4". It is encoded in UTF-8 as the hex sequence: "E2 82 AC". Those 3 bytes are contained in the file "utf8euro.txt". This command will generate the corresponding code:

```
zint -b 71 --scale=10 --eci=17 -i utf8euro.txt
```

This is equivalent to the commands (using the `--esc` switch):

```
zint -b 71 --scale=10 --eci=17 --esc -d "\xE2\x82\xAC"
```

```
zint -b 71 --scale=10 --eci=17 --esc -d "\u20AC"
```

and to the command:

```
zint -b 71 --scale=10 --eci=17 -d "€"
```



Figure 18: `zint -b DATAMATRIX --eci=17 -d "€"`

#### 4.10.2.2 Input Modes and ECI Example 2

The Chinese character with the Unicode codepoint U+5E38 can be encoded in Big5 encoding. The Big5 representation of this character is the two hex bytes: "B1 60" (contained in the file "big5char.txt"). The generation command for Data Matrix is:

<sup>4</sup>ISO/IEC 646 Invariant is a subset of ASCII with 12 characters undefined: #, \$, @, [, \, ], ^, ` , {, |, }, ~.

```
zint -b 71 --scale=10 --eci=28 --binary -i big5char.txt
```

This is equivalent to the command (using the --esc switch):

```
zint -b 71 --scale=10 --eci=28 --binary --esc -d "\xB1\x60"
```

and to the commands (no --binary switch so conversion occurs):

```
zint -b 71 --scale=10 --eci=28 --esc -d "\xE5\xB8\xB8"
```

```
zint -b 71 --scale=10 --eci=28 --esc -d "\u5E38"
```

```
zint -b 71 --scale=10 --eci=28 -d " 常"
```



Figure 19: `zint -b DATAMATRIX --eci=28 -d "\u5E38" --esc`

#### 4.10.2.3 Input Modes and ECI Example 3

Some decoders (in particular mobile app ones) for QR Code assume UTF-8 encoding by default and do not support ECI. In this case supply UTF-8 data and use the --binary switch so that the data will be encoded as UTF-8 without conversion:

```
zint -b 58 --binary -d "UTF-8 data"
```



Figure 20: `zint -b QRCODE --binary -d "\xE2\x82\xAC\xE5\xB8\xB8" --esc`

### 4.11 Batch Processing

Data can be batch processed by reading from a text file and producing a separate barcode image for each line of text in that file. To do this use the --batch switch. To select the input file from which to read data use the -i option. Zint will automatically detect the end of a line of text (in either Unix or Windows formatted text files) and produce a symbol each time it finds this. Input files should end with a line feed character - if this is not present then Zint will not encode the last line of text, and will warn you that there is a problem.

By default Zint will output numbered filenames starting with 00001.png, 00002.png etc. To change this behaviour use the -o option in combination with --batch using special characters in the output filename as shown in the table below:

Table : Batch Filename Formatting

Input Character	Interpretation
~	Insert a number or 0
#	Insert a number or space
@	Insert a number or *
Any other	Insert literally

The following table shows some examples to clarify this method:

Table : Batch Filename Examples

Input	Filenames Generated
-o file~~~.svg	file001.svg, file002.svg, file003.svg
-o @@@@bar.png	***1.png, ***2.png, ***3.png

Input	Filenames Generated
-o my~~~bar.eps	my001.bar.eps, my002.bar.eps, my003bar.eps
-o t@es~t~.png	t*es0t1.png, t*es0t2.png, t*es0t3.png

## 4.12 Direct Output

The finished image files can be output directly to stdout for use as part of a pipe by using the `--direct` option. By default `--direct` will output data as a PNG image (or GIF image if `libpng` is not present), but this can be altered by supplementing the `--direct` option with a `--filetype` option followed by the suffix of the file type required. For example:

```
zint -b 84 --direct --filetype=pcx -d "Data to encode"
```

This command will output the symbol as a PCX file to stdout. The currently supported output file formats are shown in the following table:

Table : Output File Formats

Abbreviation	File format
BMP	Windows Bitmap
EMF	Enhanced Metafile Format
EPS	Encapsulated PostScript
GIF	Graphics Interchange Format
PCX	ZSoft Paintbrush image
PNG	Portable Network Graphic
SVG	Scalable Vector Graphic
TIF	Tagged Image File Format
TXT	Text file (see <a href="#">4.18 Other Output Options</a> )

CAUTION: Outputting binary files to the command shell without catching that data in a pipe can have unpredictable results. Use with care!

## 4.13 Automatic Filenames

The `--mirror` option instructs Zint to use the data to be encoded as an indicator of the filename to be used. This is particularly useful if you are processing batch data. For example the input data "1234567" will result in a file named "1234567.png".

There are restrictions, however, on what characters can be stored in a filename, so the filename may vary from the data if the data includes non-printable characters, for example, and may be shortened if the data input is long.

To set the output file format use the `--filetype` option as detailed above in [4.12 Direct Output](#).

## 4.14 Working with Dots

Matrix codes can be rendered as a series of dots or circles rather than the normal squares by using the `--dotty` option. This option is only available for matrix symbologies, and is automatically selected for DotCode. The size of the dots can be adjusted using the `--dotsize` option followed by the diameter of the dot, where that diameter is given as a multiple of the X-dimension. The minimum dot size is 0.01, the maximum is 20. The default size is 0.8.

The default and minimum scale for raster output in dotty mode is 1.



Figure 21: `zint -b CODEONE -d "123456789012345678" --dotty --vers=9`

## 4.15 Multiple Segments

If you need to specify different ECIs for different sections of the input data, the `--seg1` to `--seg9` options can be used. Each option is of the form `--segN=ECI, data` where ECI is the ECI code (see Table : [ECI Codes](#)) and data is the data to which this applies. This is in addition to the ECI and data specified using the `--eci` and `-d` options which must still be present and which in effect constitute segment 0. For instance

```
zint -b AZTEC_CODE --eci=9 -d "Κείμενο" --seg1=7, "Текст" --seg2=20, "文章"
```

specifies 3 segments: segment 0 with ECI 9 (Greek), segment 1 with ECI 7 (Cyrillic), and segment 2 with ECI 20 (Shift JIS). Segments must be consecutive.

The symbology must be ECI-aware (see Table : [ECI-Aware Symbologies](#)).



Figure 22: `zint -b AZTEC --eci=9 -d "Κείμενο" --seg1=7, "Текст" --seg2=20, "文章"`

ECIs of zero may be given, in which case Zint will automatically determine an ECI if necessary, as described in section [4.10.2 Input Modes and ECI](#).

Multiple segments are not currently supported for use with GS1 data.

## 4.16 Structured Append

Structured Append is a method of splitting data among several symbols so that they form a sequence that can be scanned and re-assembled in the correct order on reading, and is available for Aztec Code, Code One, Data Matrix, DotCode, Grid Matrix, MaxiCode, MicroPDF417, PDF417, QR Code and Ultracode.

The `--structapp` option marks a symbol as part of a Structured Append sequence, and has the format

```
--structapp=I, C[, ID]
```



Figure 23: `zint -b DATAMATRIX -d "2nd of 3" --structapp="2,3,5006"`

where *I* is the index (position) of the symbol in the Structured Append sequence, *C* is the count or total number of symbols in the sequence, and *ID* is an optional identifier (not available for Code One, DotCode or MaxiCode) that is the same for all symbols belonging to the same sequence. The index is 1-based and goes from 1 to count. Count must be 2 or more. See the individual symbologies for further details.

## 4.17 Help Options

There are three help options which give information about how to use the command line. The `-h` or `--help` option will display a list of all of the valid options available, and also gives the exact version of the software (the version by itself can be displayed with `-v` or `--version`).

The `-t` or `--types` option gives the table of symbologies along with the symbol ID numbers and names.

The `-e` or `--ecinos` option gives a list of the ECI codes.

## 4.18 Other Output Options

For linear barcodes the text present in the output image can be removed by using the `--notext` option.

The text can be set to bold using the `--bold` option, or a smaller font can be substituted using the `--small` option. The `--bold` and `--small` options can be used together if required, but only for vector output.

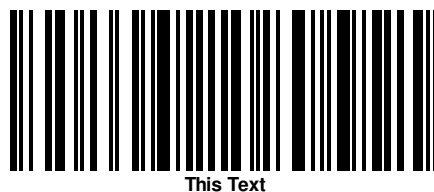


Figure 24: `zint --bold -d "This Text" --small`

Zint can output a representation of the symbol data as a set of hexadecimal values if asked to output to a text file ("`*.txt`") or if given the option `--filetype=txt`. This can be used for test and diagnostic purposes.

The `--cmyk` option is specific to output in Encapsulated PostScript and TIF, and converts the RGB colours used to the CMYK colour space. Setting custom colours at the command line will still need to be done in RRGGBB format.

Additional options are available which are specific to certain symbologies. These may, for example, control the amount of error correction data or the size of the symbol. These options are discussed in section 6. **Types of Symbology** of this guide.

## 5. Using the API

Zint has been written using the C language and has an API for use with C/C++ language programs. A Qt interface is available in the "backend\_qt" sub-directory, and a Tcl interface is available in the "backend\_tcl" sub-directory.

The libzint API has been designed to be very similar to that used by the GNU Barcode package. This allows easy migration from GNU Barcode to Zint. Zint, however, uses none of the same function names or option names as GNU Barcode. This allows you to use both packages in your application without conflict if you wish.

### 5.1 Creating and Deleting Symbols

The symbols manipulated by Zint are held in a `zint_symbol` structure defined in "zint.h". These symbols are created with the `ZBarcode_Create()` function and deleted using the `ZBarcode_Delete()` function. For example the following code creates and then deletes a symbol:

```
#include <zint.h>
#include <stdio.h>
int main()
{
    struct zint_symbol *my_symbol;
    my_symbol = ZBarcode_Create();
    if (my_symbol != NULL) {
        printf("Symbol successfully created!\n");
    }
    ZBarcode_Delete(my_symbol);
    return 0;
}
```

When compiling this code it will need to be linked with the `libzint` library using the `-lzint` option:

```
gcc -o simple simple.c -lzint
```

### 5.2 Encoding and Saving to File

To encode data in a barcode use the `ZBarcode_Encode()` function. To write the symbol to a file use the `ZBarcode_Print()` function. For example the following code takes a string from the command line and outputs a Code 128 symbol in a PNG file named "out.png" (or a GIF file called "out.gif" if `libpng` is not present) in the current working directory:

```
#include <zint.h>
int main(int argc, char **argv)
{
    struct zint_symbol *my_symbol;
    my_symbol = ZBarcode_Create();
    ZBarcode_Encode(my_symbol, argv[1], 0);
    ZBarcode_Print(my_symbol, 0);
    ZBarcode_Delete(my_symbol);
    return 0;
}
```

This can also be done in one stage using the `ZBarcode_Encode_and_Print()` function as shown in the next example:

```
#include <zint.h>
int main(int argc, char **argv)
{
    struct zint_symbol *my_symbol;
    my_symbol = ZBarcode_Create();
    ZBarcode_Encode_and_Print(my_symbol, argv[1], 0, 0);
    ZBarcode_Delete(my_symbol);
    return 0;
}
```

Note that when using the API, the input data is assumed to be 8-bit binary unless the `input_mode` variable in the `zint_symbol` structure is set - see [5.10 Setting the Input Mode](#) for details.

### 5.3 Encoding and Printing Functions in Depth

The functions for encoding and printing barcodes are defined as:

```
int ZBarcode_Encode(struct zint_symbol *symbol,
    const unsigned char *source, int length);

int ZBarcode_Encode_File(struct zint_symbol *symbol,
    const char *filename);

int ZBarcode_Print(struct zint_symbol *symbol, int rotate_angle);

int ZBarcode_Encode_and_Print(struct zint_symbol *symbol,
    const unsigned char *source, int length, int rotate_angle);

int ZBarcode_Encode_File_and_Print(struct zint_symbol *symbol,
    const char *filename, int rotate_angle);
```

In these definitions `length` can be used to set the length of the input string. This allows the encoding of NUL (ASCII 0) characters in those symbologies which allow this. A value of 0 will disable this usage and Zint will encode data up to the first NUL character in the input string, which must be present.

The `rotate_angle` value can be used to rotate the image when outputting. Valid values are 0, 90, 180 and 270.

The `ZBarcode_Encode_File()` and `ZBarcode_Encode_File_and_Print()` functions can be used to encode data read directly from a text file where the filename is given in the NUL-terminated `filename` string.

If printing more than one barcode, the `zint_symbol` structure may be re-used by calling the `ZBarcode_Clear()` function after each barcode to free any output buffers allocated. The `zint_symbol` input variables must be reset.

### 5.4 Buffering Symbols in Memory (raster)

In addition to saving barcode images to file Zint allows you to access a representation of the resulting bitmap image in memory. The following functions allow you to do this:

```
int ZBarcode_Buffer(struct zint_symbol *symbol, int rotate_angle);

int ZBarcode_Encode_and_Buffer(struct zint_symbol *symbol,
    const unsigned char *source, int length, int rotate_angle);

int ZBarcode_Encode_File_and_Buffer(struct zint_symbol *symbol,
    const char *filename, int rotate_angle);
```

The arguments here are the same as above. The difference is that instead of saving the image to a file it is placed in an unsigned character array. The bitmap pointer is set to the first memory location in the array and the values `barcode_width` and `barcode_height` indicate the size of the resulting image in pixels. Rotation and colour options can be used with the buffer functions in the same way as when saving to a file. The pixel data can be extracted from the array by the method shown in the example below where `render_pixel()` is assumed to be a function for drawing a pixel on the screen implemented by the external application:

```
int row, col, i = 0;
int red, blue, green;

for (row = 0; row < my_symbol->bitmap_height; row++) {
    for (col = 0; col < my_symbol->bitmap_width; col++) {
        red = (int) my_symbol->bitmap[i];
        green = (int) my_symbol->bitmap[i + 1];
        blue = (int) my_symbol->bitmap[i + 2];
        render_pixel(row, col, red, green, blue);
        i += 3;
    }
}
```

```

    }
}

```

Where speed is important, the buffer can be returned instead in a more compact intermediate form using the output option `OUT_BUFFER_INTERMEDIATE`. Here each byte is an ASCII value: '1' for foreground colour and '0' for background colour, except for Ultracode, which also uses colour codes: 'W' for white, 'C' for cyan, 'B' for blue, 'M' for magenta, 'R' for red, 'Y' for yellow, 'G' for green, and 'K' for black. The loop for accessing the data is then:

```

int row, col, i = 0;

for (row = 0; row < my_symbol->bitmap_height; row++) {
    for (col = 0; col < my_symbol->bitmap_width; col++) {
        render_pixel(row, col, my_symbol->bitmap[i]);
        i++;
    }
}

```

## 5.5 Buffering Symbols in Memory (vector)

Symbols can also be saved to memory in a vector representation as well as a bitmap one. The following functions, exactly analogous to the ones above, allow you to do this:

```

int ZBarcode_Buffer_Vector(struct zint_symbol *symbol, int rotate_angle);

int ZBarcode_Encode_and_Buffer_Vector(struct zint_symbol *symbol,
    const unsigned char *source, int length, int rotate_angle);

int ZBarcode_Encode_File_and_Buffer_Vector(struct zint_symbol *symbol,
    const char *filename, int rotate_angle);

```

Here the vector pointer is set to a header which contains pointers to lists of structures representing the various elements of the barcode: rectangles, hexagons, strings and circles. To draw the barcode, each of the element types is iterated in turn, and using the information stored is drawn by a rendering system. For instance, to draw a barcode using a rendering system with `prepare_canvas()`, `draw_rect()`, `draw_hexagon()`, `draw_string()`, and `draw_circle()` routines available:

```

struct zint_vector_rect *rect;
struct zint_vector_hexagon *hexagon;
struct zint_vector_string *string;
struct zint_vector_circle *circle;

prepare_canvas(my_symbol->vector->width, my_symbol->vector->height,
    my_symbol->scale, my_symbol->fgcolour, my_symbol->bcolor,
    rotate_angle);

for (rect = my_symbol->vector->rectangles; rect; rect = rect->next) {
    draw_rect(rect->x, rect->y, rect->width, rect->height,
        rect->colour);
}
for (hexagon = my_symbol->vector->hexagons; hexagon; hexagon = hexagon->next) {
    draw_hexagon(hexagon->x, hexagon->y, hexagon->diameter,
        hexagon->rotation);
}
for (string = my_symbol->vector->strings; string; string = string->next) {
    draw_string(string->x, string->y, string->fsize,
        string->rotation, string->halign,
        string->text, string->length);
}
for (circle = my_symbol->vector->circles; circle; circle = circle->next) {
    draw_circle(circle->x, circle->y, circle->diameter,
        circle->width, circle->colour);
}

```



```
}
```

## 5.6 Setting Options

So far our application is not very useful unless we plan to only make Code 128 symbols and we don't mind that they only save to "out.png". As with the CLI program, of course, these options can be altered. The way this is done is by altering the contents of the `zint_symbol` structure between the creation and encoding stages. The `zint_symbol` structure consists of the following variables:

Table : API Structure `zint_symbol`

Variable Name	Type	Meaning	Default Value
<code>symbology</code>	integer	Symbol to use (see <a href="#">5.8 Specifying a Symbology</a> ).	<code>BARCODE_CODE128</code>
<code>height</code>	float	Symbol height, excluding fixed width-to-height symbols. <sup>5</sup>	Symbol dependent
<code>scale</code>	float	Scale factor for adjusting size of image.	1.0
<code>whitespace_width</code>	integer	Horizontal whitespace width.	0
<code>whitespace_height</code>	integer	Vertical whitespace height.	0
<code>border_width</code>	integer	Border width.	0
<code>output_options</code>	integer	Set various output file parameters (see <a href="#">5.9 Adjusting Other Output Options</a> ).	0 (none)
<code>fgcolour</code>	character string	Foreground (ink) colour as RGB/RGBA hexadecimal string. Must be 6 or 8 characters followed by a terminating NUL.	"000000"
<code>bgcolour</code>	character string	Background (paper) colour as RGB/RGBA hexadecimal string. Must be 6 or 8 characters followed by a terminating NUL.	"ffffff"
<code>fgcolor</code>	pointer	Points to <code>fgcolour</code> allowing alternate spelling.	
<code>bgcolor</code>	pointer	Points to <code>bgcolour</code> allowing alternate spelling.	
<code>outfile</code>	character string	Contains the name of the file to output a resulting barcode symbol to. Must end in <code>.png</code> , <code>.gif</code> , <code>.bmp</code> , <code>.emf</code> , <code>.eps</code> , <code>.pcx</code> , <code>.svg</code> , <code>.tif</code> or <code>.txt</code> followed by a terminating NUL.	"out.png"
<code>primary</code>	character string	Primary message data for more complex symbols, with a terminating NUL.	"" (empty)
<code>option_1</code>	integer	Symbol specific options.	-1
<code>option_2</code>	integer	Symbol specific options.	0
<code>option_3</code>	integer	Symbol specific options.	0
<code>show_hrt</code>	integer	Set to 0 to hide text	1
<code>input_mode</code>	integer	Set encoding of input data (see <a href="#">5.10 Setting the Input Mode</a> ).	<code>DATA_MODE</code>
<code>eci</code>	integer	Extended Channel Interpretation code.	0 (none)
<code>dot_size</code>	float	Diameter of dots used in dotty mode.	4.0 / 5.0
<code>guard_descent</code>	float	Height of guard bar descent (UPC/EAN only)	5.0

<sup>5</sup>This value is ignored for Aztec (including HIBC and Aztec Rune), Code One, Data Matrix (including HIBC), DotCode, Grid Matrix, Han Xin, MaxiCode, QR Code (including HIBC, Micro QR, rMQR and UPNQR), and Ultracode - all of which have a fixed width-to-height ratio (or, in the case of Code One, a fixed height).

Variable Name	Type	Meaning	Default Value
structapp	Structured Append structure	Mark a symbol as part of a sequence of symbols.	count 0 (disabled)
warn_level	integer	Affects error/warning value returned by Zint API (see <a href="#">5.7 Handling Errors</a> ).	WARN_DEFAULT
text	unsigned character string	Human Readable Text, which usually consists of input data plus one more check digit. Uses UTF-8 formatting, with a terminating NUL.	"" (empty) (output only)
rows	integer	Number of rows used by the symbol.	(output only)
width	integer	Width of the generated symbol.	(output only)
encoding_data	array of unsigned character arrays	Representation of the encoded data.	(output only)
row_height	array of floats	Representation of the height of a row.	(output only)
errtxt	character string	Error message in the event that an error occurred, with a terminating NUL.	(output only)
bitmap	pointer to unsigned character array	Pointer to stored bitmap image.	(output only)
bitmap_width	integer	Width of stored bitmap image (in pixels).	(output only)
bitmap_height	integer	Height of stored bitmap image (in pixels).	(output only)
alphamap	pointer to unsigned character array	Pointer to array representing alpha channel (or NULL if no alpha channel needed)	(output only)
bitmap_byte_length	integer	Size of BMP bitmap data.	(output only)
vector	pointer to vector structure	Pointer to vector header containing pointers to vector elements.	(output only)

To alter these values use the syntax shown in the example below. This code has the same result as the previous example except the output is now taller and plotted in green.

```
#include <zint.h>
#include <string.h>
int main(int argc, char **argv)
{
    struct zint_symbol *my_symbol;
    my_symbol = ZBarcode_Create();
    strcpy(my_symbol->fgcolour, "00ff00");
    my_symbol->height = 400.0f;
    ZBarcode_Encode_and_Print(my_symbol, argv[1], 0, 0);
    ZBarcode_Delete(my_symbol);
    return 0;
}
```

Background removal for EMF, EPS, GIF, PNG, SVG and TIF files can be achieved by setting the background alpha to "00" where the values for R, G and B will be ignored:

```
strcpy(my_symbol->bgcolour, "55555500");
```

## 5.7 Handling Errors

If errors occur during encoding a non-zero integer value is passed back to the calling application. In addition the `errtxt` variable is used to give a message detailing the nature of the error. The errors generated by Zint are given in the table below:

Table : API Warning and Error Return Values

Return Value	Meaning
ZINT_WARN_INVALID_OPTION	One of the values in <code>zint_struct</code> was set incorrectly but Zint has made a guess at what it should have been and generated a barcode accordingly.
ZINT_WARN_USES_ECI	Zint has automatically inserted an ECI character. The symbol may not be readable with some readers.
ZINT_WARN_NONCOMPLIANT	The symbol was created but is not compliant with certain standards set in its specification (e.g. height, GS1 AI data lengths).
ZINT_ERROR	Marks the divide between warnings and errors. For return values greater than or equal to this no symbol (or only an incomplete symbol) is generated.
ZINT_ERROR_TOO_LONG	The input data is too long or too short for the selected symbology. No symbol has been generated.
ZINT_ERROR_INVALID_DATA	The data to be encoded includes characters which are not permitted by the selected symbology (e.g. alphabetic characters in an EAN symbol). No symbol has been generated.
ZINT_ERROR_INVALID_CHECK	Data with an incorrect check digit has been entered. No symbol has been generated.
ZINT_ERROR_INVALID_OPTION	One of the values in <code>zint_struct</code> was set incorrectly and Zint was unable to guess what it should have been. No symbol has been generated.
ZINT_ERROR_ENCODING_PROBLEM	A problem has occurred during encoding of the data. This should never happen. Please contact the developer if you encounter this error.
ZINT_ERROR_FILE_ACCESS	Zint was unable to open the requested output file. This is usually a file permissions problem.
ZINT_ERROR_MEMORY	Zint ran out of memory. This should only be a problem with legacy systems.
ZINT_ERROR_FILE_WRITE	Zint failed to write all contents to the requested output file. This should only occur if the output device becomes full.
ZINT_ERROR_USES_ECI	Returned if <code>warn_level</code> set to <code>WARN_FAIL_ALL</code> and <code>ZINT_WARN_USES_ECI</code> occurs.
ZINT_ERROR_NONCOMPLIANT	Returned if <code>warn_level</code> set to <code>WARN_FAIL_ALL</code> and <code>ZINT_WARN_NONCOMPLIANT</code> occurs.

To catch errors use an integer variable as shown in the code below:

```
#include <zint.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv)
{
    struct zint_symbol *my_symbol;
    int error;
    my_symbol = ZBarcode_Create();
    strcpy(my_symbol->fgcolour, "nonsense");
    error = ZBarcode_Encode_and_Print(my_symbol, argv[1], 0, 0);
    if (error != 0) {
```

```

    /* some warning or error occurred */
    printf("%s\n", my_symbol->errtxt);
}
if (error >= ZINT_ERROR) {
    /* stop now */
    ZBarcode_Delete(my_symbol);
    return 1;
}
/* otherwise carry on with the rest of the application */
ZBarcode_Delete(my_symbol);
return 0;
}

```

This code will exit with the appropriate message:

Error 653: Malformed foreground colour 'NONSENSE' (hexadecimal only)

To treat all warnings as errors, set `symbol->warn_level` to `WARN_FAIL_ALL`.

## 5.8 Specifying a Symbology

Symbologies can be specified by number or by name as shown in the Table : : **Barcode Types (Symbologies)**. For example

```
symbol->symbology = BARCODE_LOGMARS;
```

means the same as

```
symbol->symbology = 50;
```

## 5.9 Adjusting Other Output Options

The `output_options` variable can be used to adjust various aspects of the output file. To select more than one option from the table below simply OR them together when adjusting this value:

```
my_symbol->output_options |= BARCODE_BIND | READER_INIT;
```

Table : API `output_options` Values

Value	Effect
0	No options selected.
BARCODE_BIND	Boundary bars above and below the symbol and between rows if stacking multiple symbols. <sup>6</sup>
BARCODE_BOX	Add a box surrounding the symbol and whitespace.
BARCODE_STDOUT	Output the file to stdout.
READER_INIT	Add a reader initialisation symbol to the data before encoding.
SMALL_TEXT	Use a smaller font for the Human Readable Text.
BOLD_TEXT	Embolden the Human Readable Text.
CMYK_COLOUR	Select the CMYK colour space option for Encapsulated PostScript and TIF files.
BARCODE_DOTTY_MODE	Plot a matrix symbol using dots rather than squares.
GS1_GS_SEPARATOR	Use GS instead of FNC1 as GS1 separator (Data Matrix only).
OUT_BUFFER_INTERMEDIATE	Return the bitmap buffer as ASCII values instead of separate colour channels (OUT_BUFFER only).
BARCODE_QUIET_ZONES	Add compliant quiet zones (additional to any specified whitespace). <sup>7</sup>
BARCODE_NO_QUIET_ZONES	Disable quiet zones, notably those with defaults.
COMPLIANT_HEIGHT	Warn if height not compliant and use standard height (if any) as default.

<sup>6</sup>This flag is always set for Codablock-F, Code 16K and Code 49. Special considerations apply to ITF-14 - see [6.1.2.6 ITF-14](#).

<sup>7</sup>Codablock-F, Code 16K, Code 49, ITF-14, EAN-2 to EAN-13, ISBN, UPC-A and UPC-E have compliant quiet zones added by default.

## 5.10 Setting the Input Mode

The way in which the input data is encoded can be set using the `input_mode` property. Valid values are shown in the table below.

Table : API `input_mode` Values

Value	Effect
DATA_MODE	Uses full 8-bit range interpreted as binary data.
UNICODE_MODE	Uses UTF-8 input.
GS1_MODE	Encodes GS1 data using FNC1 characters. <i>The above are exclusive, the following optional and OR-ed.</i>
ESCAPE_MODE	Process input data for escape sequences.
GS1PARENS_MODE	Parentheses (round brackets) used in GS1 data instead of square brackets to delimit Application Identifiers (parentheses must not otherwise occur in the data).
GS1NOCHECK_MODE	Do not check GS1 data for validity, i.e. suppress checks for valid AIs and data lengths. Invalid characters (e.g. control characters, extended ASCII characters) are still checked for.
HEIGHTPERROW_MODE	Interpret the <code>height</code> variable as per-row rather than as overall height.
FAST_MODE	Use faster if less optimal encodation for symbologies that support it (currently DATAMATRIX only).

The default mode is `DATA_MODE`. (Note that this differs from the default for the CLI and GUI, which is `UNICODE_MODE`.)

`DATA_MODE`, `UNICODE_MODE` and `GS1_MODE` are mutually exclusive, whereas `ESCAPE_MODE`, `GS1PARENS_MODE`, `GS1NOCHECK_MODE`, `HEIGHTPERROW_MODE` and `FAST_MODE` are optional. So, for example, you can set

```
my_symbol->input_mode = UNICODE_MODE | ESCAPE_MODE;
```

or

```
my_symbol->input_mode = GS1_MODE | GS1PARENS_MODE | GS1NOCHECK_MODE;
```

whereas

```
my_symbol->input_mode = DATA_MODE | GS1_MODE;
```

is not valid.

Permissible escape sequences are listed in Table : [Escape Sequences](#). An example of `GS1PARENS_MODE` usage is given in section [6.1.10.3 GS1-128](#).

`GS1NOCHECK_MODE` is for use with legacy systems that have data that does not conform to the current GS1 standard. Printable ASCII input is still checked for, as is the validity of GS1 data specified without AIs (e.g. linear data for GS1 DataBar Omnidirectional/Limited/etc.).

For `HEIGHTPERROW_MODE`, see `--heightperrow` in section [4.4 Adjusting Height](#). The `height` variable should be set to the desired per-row value on input (it will be set to the overall height on output).

## 5.11 Multiple Segments

For input data requiring multiple ECIs, the following functions may be used:

```
int ZBarcode_Encode_Segs(struct zint_symbol *symbol,
    const struct zint_seg segs[], const int seg_count);

int ZBarcode_Encode_Segs_and_Print(struct zint_symbol *symbol,
    const struct zint_seg segs[], const int seg_count, int rotate_angle);

int ZBarcode_Encode_Segs_and_Buffer(struct zint_symbol *symbol,
    const struct zint_seg segs[], const int seg_count, int rotate_angle);
```

```
int ZBarcode_Encode_Segs_and_Buffer_Vector(struct zint_symbol *symbol,
    const struct zint_seg segs[], const int seg_count, int rotate_angle);
```

These are direct analogues of the previously mentioned `ZBarcode_Encode()`, `ZBarcode_Encode_and_Print()`, `ZBarcode_Encode_and_Buffer()` and `ZBarcode_Encode_and_Buffer_Vector()` respectively, where instead of a pair consisting of "source, length", a pair consisting of "segs, seg\_count" is given, with segs being an array of `struct zint_seg` segments and seg\_count being the number of elements it contains. The `zint_seg` structure is of the form:

```
struct zint_seg {
    unsigned char *source; /* Data to encode */
    int length;           /* Length of `source`. If 0, `source` must be
                           NUL-terminated */
    int eci;              /* Extended Channel Interpretation */
};
```

The symbology must support ECIs (see Table : [ECI-Aware Symbologies](#)). For example:

```
#include <zint.h>
int main(int argc, char **argv)
{
    struct zint_seg segs[] = {
        { "Κείμενο", 0, 9 },
        { "Текст", 0, 7 },
        { " 文章", 0, 20 }
    };
    struct zint_symbol *my_symbol;
    my_symbol = ZBarcode_Create();
    my_symbol->symbology = BARCODE_AZTEC;
    my_symbol->input_mode = UNICODE_MODE;
    ZBarcode_Encode_Segs(my_symbol, segs, 3);
    ZBarcode_Print(my_symbol, 0);
    ZBarcode_Delete(my_symbol);
    return 0;
}
```

A maximum of 256 segments may be specified. Use of multiple segments with GS1 data is not currently supported.

## 5.12 Verifying Symbology Availability

An additional function available in the API is:

```
int ZBarcode_ValidID(int symbol_id);
```

which allows you to check whether a given symbology is available, returning a non-zero value if so. For example:

```
if (ZBarcode_ValidID(BARCODE_PDF417) != 0) {
    printf("PDF417 available\n");
} else {
    printf("PDF417 not available\n");
}
```

Another function that may be useful is:

```
int ZBarcode_BarcodeName(int symbol_id, char name[32]);
```

which copies the name of a symbology into the supplied name buffer, which should be 32 characters in length. The name is NUL-terminated, and zero is returned on success. For instance:

```
char name[32];
if (ZBarcode_BarcodeName(BARCODE_PDF417, name) == 0) {
    printf("%s\n", name);
}
```

will print `BARCODE_PDF417`.

### 5.13 Checking Symbology Capabilities

It can be useful for frontend programs to know the capabilities of a symbology. This can be determined using another additional function:

```
unsigned int ZBarcode_Cap(int symbol_id, unsigned int cap_flag);
```

by OR-ing the flags below in the `cap_flag` argument and checking the return to see which are set.

Table : API Capability Flags

Value	Meaning
ZINT_CAP_HRT	Can the symbology print Human Readable Text?
ZINT_CAP_STACKABLE	Is the symbology stackable?
ZINT_CAP_EXTENDABLE	Is the symbology extendable with add-on data? (i.e. is it UPC/EAN?)
ZINT_CAP_COMPOSITE	Does the symbology support composite data? (see <a href="#">6.3 Composite Symbols (ISO 24723)</a> below)
ZINT_CAP_ECI	Does the symbology support Extended Channel Interpretations?
ZINT_CAP_GS1	Does the symbology support GS1 data?
ZINT_CAP_DOTTY	Can the symbology be outputted as dots?
ZINT_CAP_QUIET_ZONES	Does the symbology have default quiet zones?
ZINT_CAP_FIXED_RATIO	Does the symbology have a fixed width-to-height (aspect) ratio?
ZINT_CAP_READER_INIT	Does the symbology support Reader Initialisation?
ZINT_CAP_FULL_MULTIBYTE	Is the ZINT_FULL_MULTIBYTE option applicable?
ZINT_CAP_MASK	Is mask selection applicable?
ZINT_CAP_STRUCTAPP	Does the symbology support Structured Append?
ZINT_CAP_COMPLIANT_HEIGHT	Does the symbology have a compliant height defined?

For example:

```
unsigned int cap = ZBarcode_Cap(BARCODE_PDF417, ZINT_CAP_HRT | ZINT_CAP_ECI);
if (cap & ZINT_CAP_HRT) {
    printf("PDF417 supports HRT\n");
} else {
    printf("PDF417 does not support HRT\n");
}
if (cap & ZINT_CAP_ECI) {
    printf("PDF417 supports ECI\n");
} else {
    printf("PDF417 does not support ECI\n");
}
```

### 5.14 Zint Version

Lastly, the version of the Zint library linked to is returned by:

```
int ZBarcode_Version();
```

The version parts are separated by hundreds. For instance, version "2.9.1" is returned as "20901".

## 6. Types of Symbology

### 6.1 One-Dimensional Symbols

One-dimensional or linear symbols are what most people associate with the term barcode. They consist of a number of bars and a number of spaces of differing widths.

#### 6.1.1 Code 11



Figure 25: `zint -b CODE11 -d "9212320967"`

Developed by Intermec in 1977, Code 11 is similar to Code 2 of 5 Matrix and is primarily used in telecommunications. The symbol can encode data consisting of the digits 0-9 and the dash character (-) up to a maximum of 121 characters. Two modulo-11 check digits are added by default. To add just one check digit, set `--vers=1` (API `option_2 = 1`). To add no check digits, set `--vers=2` (API `option_2 = 2`).

#### 6.1.2 Code 2 of 5

Code 2 of 5 is a family of one-dimensional symbols, 8 of which are supported by Zint. Note that the names given to these standards alters from one source to another so you should take care to ensure that you have the right barcode type before using these standards.

##### 6.1.2.1 Standard Code 2 of 5



Figure 26: `zint -b C25STANDARD -d "9212320967"`

Also known as Code 2 of 5 Matrix this is a self-checking code used in industrial applications and photo development. Standard Code 2 of 5 will encode numeric input (digits 0-9) up to a maximum of 80 digits. No check digit is added by default. To add a check digit, set `--vers=1` (API `option_2 = 1`). To add a check digit but not show it in the Human Readable Text, set `--vers=2` (API `option_2 = 2`).

##### 6.1.2.2 IATA Code 2 of 5



Figure 27: `zint -b C25IATA -d "9212320967"`

Used for baggage handling in the air-transport industry by the International Air Transport Agency, this self-checking code will encode numeric input (digits 0-9) up to a maximum of 45 digits. No check digit is added by default. To add a check digit, set `--vers=1` (API `option_2 = 1`). To add a check digit but not show it in the Human Readable Text, set `--vers=2` (API `option_2 = 2`).



### 6.1.2.3 Industrial Code 2 of 5



Figure 28: `zint -b C25IND -d "9212320967"`

Industrial Code 2 of 5 can encode numeric input (digits 0-9) up to a maximum of 45 digits. No check digit is added by default. To add a check digit, set `--vers=1` (API `option_2 = 1`). To add a check digit but not show it in the Human Readable Text, set `--vers=2` (API `option_2 = 2`).

### 6.1.2.4 Interleaved Code 2 of 5 (ISO 16390)



Figure 29: `zint -b C25INTER --compliantheight -d "9212320967"`

This self-checking symbology encodes pairs of numbers, and so can only encode an even number of digits (0-9). If an odd number of digits is entered a leading zero is added by Zint. A maximum of 45 pairs (90 digits) can be encoded. No check digit is added by default. To add a check digit, set `--vers=1` (API `option_2 = 1`). To add a check digit but not show it in the Human Readable Text, set `--vers=2` (API `option_2 = 2`).

### 6.1.2.5 Code 2 of 5 Data Logic



Figure 30: `zint -b C25LOGIC -d "9212320967"`

Data Logic does not include a check digit by default and can encode numeric input (digits 0-9) up to a maximum of 80 digits. To add a check digit, set `--vers=1` (API `option_2 = 1`). To add a check digit but not show it in the Human Readable Text, set `--vers=2` (API `option_2 = 2`).

### 6.1.2.6 ITF-14

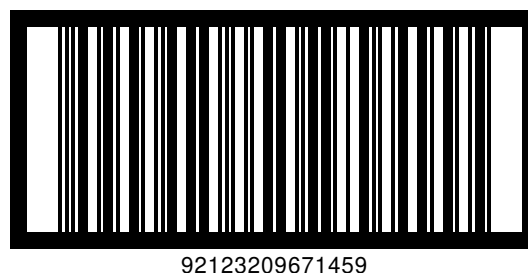


Figure 31: `zint -b ITF14 --compliantheight -d "92123209671459"`

ITF-14, also known as UPC Shipping Container Symbol or Case Code, is based on Interleaved Code 2 of 5 and requires a 13 digit numeric input (digits 0-9). One modulo-10 check digit is added by Zint.

If no border option is specified Zint defaults to adding a bounding box with a border width of 5. This behaviour can be overridden by using the `--bind` option (API `output_options |= BARCODE_BIND`). Similarly the border width can be overridden using `--border` (API `border_width`). If a symbol with no border is required this can be achieved by explicitly setting the border type to `box` (or `bind`) and leaving the border width 0.



Figure 32: `zint -b ITF14 --box --complantheight -d "9212320967145"`

#### 6.1.2.7 Deutsche Post Leitcode



Figure 33: `zint -b DPLEIT -d "9212320967145"`

Leitcode is based on Interleaved Code 2 of 5 and is used by Deutsche Post for mailing purposes. Leitcode requires a 13-digit numerical input and includes a check digit.

#### 6.1.2.8 Deutsche Post Identcode



Figure 34: `zint -b DPIDENT -d "91232096712"`

Identcode is based on Interleaved Code 2 of 5 and is used by Deutsche Post for mailing purposes. Identcode requires an 11-digit numerical input and includes a check digit.

### 6.1.3 Universal Product Code (ISO 15420)

#### 6.1.3.1 UPC Version A



Figure 35: `zint -b UPCA --complantheight -d "72527270270"`

UPC-A is used in the United States for retail applications. The symbol requires an 11 digit article number. The check digit is calculated by Zint. In addition EAN-2 and EAN-5 add-on symbols can be added using the + character. For example, to draw a UPC-A symbol with the data 72527270270 with an EAN-5 add-on showing the data 12345 use the command:

```
zint -b UPCA -d 72527270270+12345
```

or encode a data string with the + character included:

```
my_symbol->symbology = BARCODE_UPCA;
```

```
error = ZBarcode_Encode_and_Print(my_symbol, "72527270270+12345", 0, 0);
```



Figure 36: `zint -b UPCA --compliantheight -d "72527270270+12345"`

If your input data already includes the check digit symbology `BARCODE_UPCA_CHK` (35) can be used which takes a 12 digit input and validates the check digit before encoding.

You can adjust the gap between the main symbol and an add-on in multiples of the X-dimension by setting `--addongap` (API `option_2`) to a value between 9 (default) and 12. The height in X-dimensions that the guard bars descend below the main bars can be adjusted by setting `--guarddescent` (API `guard_descent`) to a value between 0 and 20 (default 5).

### 6.1.3.2 UPC Version E

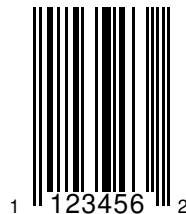


Figure 37: `zint -b UPCE --compliantheight -d "1123456"`

UPC-E is a zero-compressed version of UPC-A developed for smaller packages. The code requires a 6 digit article number (digits 0-9). The check digit is calculated by Zint. EAN-2 and EAN-5 add-on symbols can be added using the + character as with UPC-A. In addition Zint also supports Number System 1 encoding by entering a 7-digit article number stating with the digit 1. For example:

```
zint -b UPCE -d 1123456
```

or

```
my_symbol->symbology = BARCODE_UPCE;
```

```
error = ZBarcode_Encode_and_Print(my_symbol, "1123456", 0, 0);
```

If your input data already includes the check digit symbology `BARCODE_UPCE_CHK` (38) can be used which takes a 7 or 8 digit input and validates the check digit before encoding.

You can adjust the gap between the main symbol and an add-on in multiples of the X-dimension by setting `--addongap` (API `option_2`) to a value between 7 (default) and 12. The height in X-dimensions that the guard bars descend below the main bars can be adjusted by setting `--guarddescent` (API `guard_descent`) to a value between 0 and 20 (default 5).

## 6.1.4 European Article Number (ISO 15420)

### 6.1.4.1 EAN-2, EAN-5, EAN-8 and EAN-13



Figure 38: `zint -b EANX --compliantheight -d "4512345678906"`

The EAN system is used in retail across Europe and includes standards for EAN-2, EAN-5, EAN-8 and EAN-13 which encode 2, 5, 7 or 12 digit numbers respectively. Zint will decide which symbology to use depending on the length of the input data. In addition EAN-2 and EAN-5 add-on symbols can be added to EAN-8 and EAN-13 symbols using the + character as with UPC symbols. For example:

```
zint -b EANX -d 54321
```

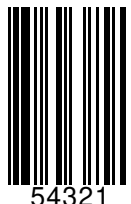


Figure 39: `zint -b EANX --compliantheight -d "54321"`

will encode a stand-alone EAN-5, whereas

```
zint -b EANX -d 7432365+54321
```

will encode an EAN-8 symbol with an EAN-5 add-on. As before these results can be achieved using the API:

```
my_symbol->symbology = BARCODE_EANX;
```

```
error = ZBarcode_Encode_and_Print(my_symbol, "54321", 0, 0);
```

```
error = ZBarcode_Encode_and_Print(my_symbol, "7432365+54321", 0, 0);
```



Figure 40: `zint -b EANX --compliantheight -d "7432365+54321"`

All of the EAN symbols include check digits which are added by Zint.

If you are encoding an EAN-8 or EAN-13 symbol and your data already includes the check digit then you can use symbology `BARCODE_EANX_CHK` (14) which takes an 8 or 13 digit input and validates the check digit before encoding.

You can adjust the gap between the main symbol and an add-on in multiples of the X-dimension by setting `--addongap` (API `option_2`) to a value between 7 (default) and 12. The height in X-dimensions that the guard bars descend below the main bars can be adjusted by setting `--guarddescent` (API `guard_descent`) to a value between 0 and 20 (default 5).

6.1.4.2 SBN, ISBN and ISBN-13



Figure 41: `zint -b ISBNX --complianceheight -d "9789295055124"`

EAN-13 symbols (also known as Bookland EAN-13) can also be produced from 9-digit SBN, 10-digit ISBN or 13-digit ISBN-13 data. The relevant check digit needs to be present in the input data and will be verified before the symbol is generated. In addition EAN-2 and EAN-5 add-on symbols can be added using the + character as with UPC symbols, and the gap set with `--addongap` (API `option_2`) to between 7 (default) and 12. The height that the guard bars descend can be adjusted by setting `--guarddescent` (API `guard_descent`) to a value between 0 and 20 (default 5).

6.1.5 Plessey  
6.1.5.1 UK Plessey



Figure 42: `zint -b PLESSEY -d "C64"`

Also known as Plessey Code, this symbology was developed by the Plessey Company Ltd. in the UK. The symbol can encode data consisting of digits (0-9) or letters A-F up to a maximum of 65 characters and includes a CRC check digit.

6.1.5.2 MSI Plessey



Figure 43: `zint -b MSI_PLESSEY -d "6502" --vers=2`

Based on Plessey and developed by MSE Data Corporation, MSI Plessey has a range of check digit options that are selectable by setting `--vers` (API `option_2`). Numeric (digits 0-9) input can be encoded, up to a maximum of 65 digits. The table below shows the options available:

Table : MSI Plessey Check Digit Options

Value	Check Digits
0	None
1	Modulo-10 (Luhn)
2	Modulo-10 & Modulo-10
3	Modulo-11 (IBM)
4	Modulo-11 (IBM) & Modulo-10

Value	Check Digits
5	Modulo-11 (NCR)
6	Modulo-11 (NCR) & Modulo-10

To not show the check digit or digits in the Human Readable Text, add 10 to the `--vers` value. For example `--vers=12` (API `option_2 = 12`) will add two hidden modulo-10 check digits.

## 6.1.6 Telepen

### 6.1.6.1 Telepen Alpha



Figure 44: `zint -b TELEPEN --compliantheight -d "Z80"`

Telepen Alpha was developed by SB Electronic Systems Limited and can encode ASCII text input, up to a maximum of 30 characters. Telepen includes a modulo-127 check digit.

### 6.1.6.2 Telepen Numeric



Figure 45: `zint -b TELEPEN_NUM --compliantheight -d "466X33"`

Telepen Numeric allows compression of numeric data into a Telepen symbol. Data can consist of pairs of numbers or pairs consisting of a numerical digit followed an X character. For example: 466333 and 466X33 are valid codes whereas 46X333 is not (the digit pair "X3" is not valid). Up to 60 digits can be encoded. Telepen Numeric includes a modulo-127 check digit which is added by Zint.

## 6.1.7 Code 39

### 6.1.7.1 Standard Code 39 (ISO 16388)

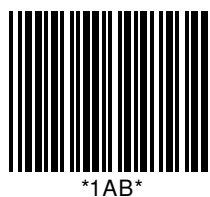


Figure 46: `zint -b CODE39 --compliantheight -d "1A" --vers=1`

Standard Code 39 was developed in 1974 by Intermec. Input data can be up to 85 characters in length and can include the characters 0-9, A-Z, dash (-), full stop (.), space, asterisk (\*), dollar (\$), slash (/), plus (+) and percent (%). The standard does not require a check digit but a modulo-43 check digit can be added if required by setting `--vers=1` (API `option_2 = 1`).

### 6.1.7.2 Extended Code 39



Figure 47: `zint -b EXCODE39 --compliantheight -d "123.45$@fd"`

Also known as Code 39e and Code39+, this symbology expands on Standard Code 39 to provide support for the full 7-bit ASCII character set. The standard does not require a check digit but a modulo-43 check digit can be added if required by setting `--vers=1` (API `option_2 = 1`).

#### 6.1.7.3 Code 93



Figure 48: `zint -b CODE93 --compliantheight -d "C93"`

A variation of Extended Code 39, Code 93 also supports full ASCII text. Two check characters are added by Zint. By default these check characters are not shown in the Human Readable Text, but may be shown by setting `--vers=1` (API `option_2 = 1`).

#### 6.1.7.4 PZN (Pharmazentralnummer)



Figure 49: `zint -b PZN --compliantheight -d "2758089"`

PZN is a Code 39 based symbology used by the pharmaceutical industry in Germany. PZN encodes a 7 digit number to which Zint will add a modulo-11 check digit.

#### 6.1.7.5 LOGMARS



Figure 50: `zint -b LOGMARS --compliantheight -d "12345/ABCDE" --vers=1`

LOGMARS (Logistics Applications of Automated Marking and Reading Symbols) is a variation of the Code 39 symbology used by the US Department of Defense. LOGMARS encodes the same character set as Standard Code 39. It does not require a check digit but a modulo-43 check digit can be added by setting `--vers=1` (API `option_2 = 1`).

#### 6.1.7.6 Code 32



Figure 51: `zint -b CODE32 --compliantheight -d "14352312"`

A variation of Code 39 used by the Italian Ministry of Health ("Ministero della Sanità") for encoding identifiers on pharmaceutical products. This symbology requires a numeric input up to 8 digits in length. A check digit is added by Zint.

#### 6.1.7.7 HIBC Code 39

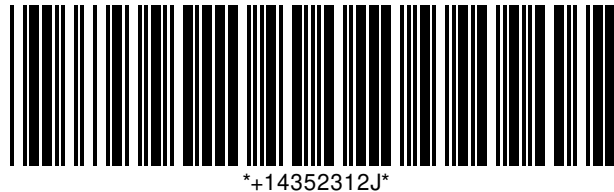


Figure 52: `zint -b HIBC_39 --compliantheight -d "14352312"`

This option adds a leading '+' character and a trailing modulo-49 check digit to a standard Code 39 symbol as required by the Health Industry Barcode standards.

#### 6.1.7.8 Vehicle Identification Number (VIN)



Figure 53: `zint -b VIN -d "2FTPX28L0XCA15511" --vers=1`

A variation of Code 39 that for vehicle identification numbers used in North America (first character '1' to '5') has a check character verification stage. A 17 character input (0-9, and A-Z excluding 'I', 'O' and 'Q') is required. An invisible Import character prefix 'I' can be added by setting `--vers=1` (API option\_2 = 1).

#### 6.1.8 Codabar (EN 798)



Figure 54: `zint -b CODABAR --compliantheight -d "A37859B"`

Also known as NW-7, Monarch, ABC Codabar, USD-4, Ames Code and Code 27, this symbology was developed in 1972 by Monarch Marketing Systems for retail purposes. The American Blood Commission adopted Codabar in 1977 as the standard symbology for blood identification. Codabar can encode up to 60 characters starting and ending with the letters A-D and containing between these letters the numbers 0-9, dash (-), dollar (\$), colon (:), slash (/), full stop (.) or plus (+). No check character is generated by default, but a modulo-16 one can be added by setting `--vers=1` (API option\_2 = 1). To have the check character appear in the Human Readable Text, set `--vers=2` (API option\_2 = 2).



### 6.1.9 Pharmacode



Figure 55: `zint -b PHARMA --compliantheight -d "130170"`

Developed by Laetus, Pharmacode is used for the identification of pharmaceuticals. The symbology is able to encode whole numbers between 3 and 131070.

### 6.1.10 Code 128

#### 6.1.10.1 Standard Code 128 (ISO 15417)

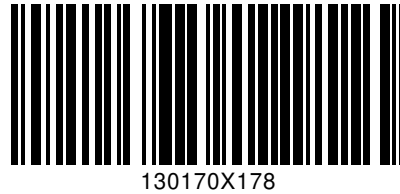


Figure 56: `zint -b CODE128 --bind -d "130170X178"`

One of the most ubiquitous one-dimensional barcode symbologies, Code 128 was developed in 1981 by Computer Identics. This symbology supports full ASCII text and uses a three-mode system to compress the data into a smaller symbol. Zint automatically switches between modes and adds a modulo-103 check digit. Code 128 is the default barcode symbology used by Zint. In addition Zint supports the encoding of ISO/IEC 8859-1 (non-English) characters in Code 128 symbols. The ISO/IEC 8859-1 character set is shown in [Appendix A.2 Latin Alphabet No. 1 \(ISO/IEC 8859-1\)](#).

#### 6.1.10.2 Code 128 Subset B



Figure 57: `zint -b CODE128B -d "130170X178"`

It is sometimes advantageous to stop Code 128 from using subset mode C which compresses numerical data. The `BARCODE_CODE128B` option (symbology 60) suppresses mode C in favour of mode B.

#### 6.1.10.3 GS1-128

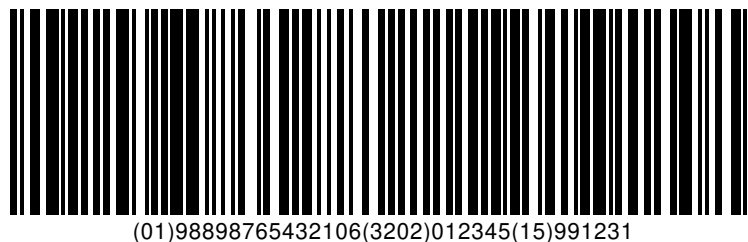


Figure 58: `zint -b GS1_128 --compliantheight -d "[01]98898765432106[3202]012345[15]991231"`

A variation of Code 128 previously known as UCC/EAN-128, this symbology is defined by the GS1 General Specifications. Application Identifiers (AIs) should be entered using [square bracket] notation. These will be

converted to parentheses (round brackets) for the Human Readable Text. This will allow round brackets to be used in the data strings to be encoded.

For compatibility with data entry in other systems, if the data does not include round brackets, the option `--gs1parens` (API `input_mode |= GS1PARENS_MODE`) may be used to signal that AIs are encased in round brackets instead of square ones.

Fixed length data should be entered at the appropriate length for correct encoding. GS1-128 does not support extended ASCII characters. Check digits for GTIN data AI (01) are not generated and need to be included in the input data. The following is an example of a valid GS1-128 input:

```
zint -b 16 -d "[01]98898765432106[3202]012345[15]991231"
```

or using the `--gs1parens` option:

```
zint -b 16 --gs1parens -d "(01)98898765432106(3202)012345(15)991231"
```

#### 6.1.10.4 EAN-14



Figure 59: `zint -b EAN14 --complantheight -d "9889876543210"`

A shorter version of GS1-128 which encodes GTIN data only. A 13 digit number is required. The GTIN check digit and AI (01) are added by Zint.

#### 6.1.10.5 NVE-18 (SSCC-18)

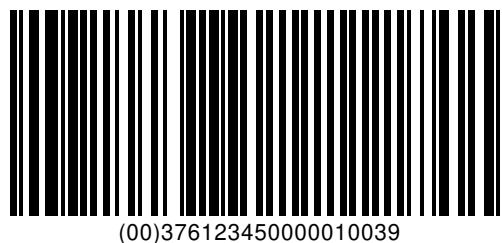


Figure 60: `zint -b NVE18 --complantheight -d "37612345000001003"`

A variation of Code 128 the 'Nummer der Versandeinheit' standard, also known as SSCC-18 (Serial Shipping Container Code), includes both modulo-10 and modulo-103 check digits. NVE-18 requires a 17 digit numerical input. Check digits and AI (00) are added by Zint.

#### 6.1.10.6 HIBC Code 128



Figure 61: `zint -b HIBC_128 -d "A123BJC5D6E71"`

This option adds a leading '+' character and a trailing modulo-49 check digit to a standard Code 128 symbol as required by the Health Industry Barcode standards.

### 6.1.10.7 DPD Code



Figure 62: `zint -b DPD --complianceheight -d "%000393206219912345678101040"`

Another variation of Code 128 as used by DPD (Deutscher Paket Dienst). Requires a 28 character alphanumeric input. Zint formats Human Readable Text as specified by DPD and adds a modulo-36 check character.

### 6.1.11 GS1 DataBar (ISO 24724)

Previously known as RSS (Reduced Spaced Symbology) these symbols are due to replace GS1-128 symbols in accordance with the GS1 General Specifications. If a GS1 DataBar symbol is to be printed with a 2D component as specified in ISO/IEC 24723 set `--mode=2` (API option\_1 = 2). See [6.3 Composite Symbols \(ISO 24723\)](#) to find out how to generate DataBar symbols with 2D components.

#### 6.1.11.1 GS1 DataBar Omnidirectional and GS1 DataBar Truncated



Figure 63: `zint -b DBAR_OMN --complianceheight -d "0950110153001"`

Previously known as RSS-14 this standard encodes a 13 digit item code. A check digit and Application Identifier of (01) are added by Zint. (A 14 digit code that appends the check digit may be given, in which case the check digit will be verified.) To produce a truncated symbol set the symbol height to a value between 13 and 32. Truncated symbols may not be scannable by omnidirectional scanners. Normal DataBar Omnidirectional symbols should have a height of 33 or greater.



Figure 64: `zint -b DBAR_OMN -d "0950110153001" --height=13`

#### 6.1.11.2 GS1 DataBar Limited



Figure 65: `zint -b DBAR_LTD --complianceheight -d "0950110153001"`

Previously known as RSS Limited this standard encodes a 13 digit item code and can be used in the same way as DataBar Omnidirectional above. DataBar Limited, however, is limited to data starting with digits 0 and 1 (i.e. numbers in the range 0 to 1999999999999). As with DataBar Omnidirectional a check digit and Application Identifier of (01) are added by Zint, and a 14 digit code may be given in which case the check digit will be verified.

6.1.11.3 GS1 DataBar Expanded



Figure 66: `zint -b DBAR_EXP --compliantheight -d "[01]98898765432106[3202]012345[15]991231"`

Previously known as RSS Expanded this is a variable length symbology capable of encoding data from a number of AIs in a single symbol. AIs should be encased in [square brackets] in the input data. This will be converted to parentheses (round brackets) before it is included in the Human Readable Text attached to the symbol. This method allows the inclusion of parentheses in the data to be encoded. If the data does not include parentheses, the AIs may alternatively be encased in parentheses using the `--gs1parens` switch. See [6.1.10.3 GS1-128](#).

GTIN data AI (01) should also include the check digit data as this is not calculated by Zint when this symbology is encoded. Fixed length data should be entered at the appropriate length for correct encoding. The following is an example of a valid DataBar Expanded input:

```
zint -b 31 -d "[01]98898765432106[3202]012345[15]991231"
```

6.1.12 Korea Post Barcode

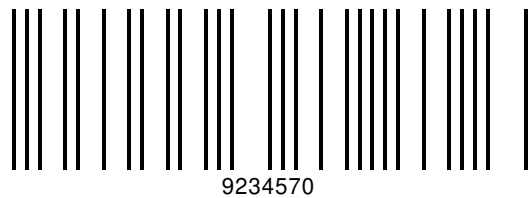


Figure 67: `zint -b KOREAPOST -d "923457"`

The Korean Postal Barcode is used to encode a six-digit number and includes one check digit.

6.1.13 Channel Code



Figure 68: `zint -b CHANNEL -d "453678" --compliantheight`

A highly compressed symbol for numeric data. The number of channels in the symbol can be between 3 and 8 and this can be specified by setting the value of the `--vers` option (API `option_2`). It can also be determined by the length of the input data e.g. a three character input string generates a 4 channel code by default.

The maximum values permitted depend on the number of channels used as shown in the table below:

Table : Channel Maximum Values

Channels	Minimum Value	Maximum Value
3	00	26
4	000	292
5	0000	3493
6	00000	44072
7	000000	576688
8	0000000	7742862

## 6.2 Stacked Symbolologies

### 6.2.1 Basic Symbol Stacking

An early innovation to get more information into a symbol, used primarily in the vehicle industry, is to simply stack one-dimensional codes on top of each other. This can be achieved at the command prompt by giving more than one set of input data. For example

```
zint -d "This" -d "That"
```

will draw two Code 128 symbols, one on top of the other. The same result can be achieved using the API by executing the `ZBarcode_Encode()` function more than once on a symbol. For example:

```
my_symbol->symbology = BARCODE_CODE128;  
  
error = ZBarcode_Encode(my_symbol, "This", 0);  
  
error = ZBarcode_Encode(my_symbol, "That", 0);  
  
error = ZBarcode_Print(my_symbol);
```

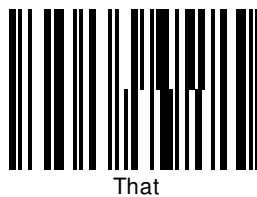


Figure 69: `zint -d "This" -d "That"`

Note that the Human Readable Text will be that of the last data, so it's best to use the option `--notext` (API `show_hrt = 0`).

The stacked barcode rows can be separated by row separator bars by specifying `--bind` (API `output_options |= BARCODE_BIND`). The height of the row separator bars in multiples of the X-dimension (minimum and default 1, maximum 4) can be set by `--separator` (API `option_3`):

```
zint --bind --notext --separator=2 -d "This" -d "That"
```

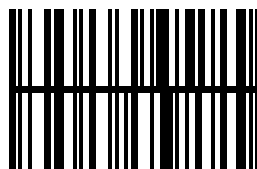


Figure 70: `zint --notext --bind --separator=2 -d "This" -d "That"`

A more sophisticated method is to use some type of line indexing which indicates to the barcode reader which order the symbols should be read. This is demonstrated by the symbologies below.

### 6.2.2 Codablock-F



Figure 71: `zint -b CODABLOCKF -d "CODABLOCK F Symbology" --rows=3`

This is a stacked symbology based on Code 128 which can encode extended ASCII code set data up to a maximum length of 2725 characters. The width of the Codablock-F symbol can be set using the `--cols` option (API `option_2`). The height (number of rows) can be set using the `--rows` option (API `option_1`). Zint does not currently support encoding of GS1 data in Codablock-F symbols.

A separate symbology ID (BARCODE\_HIBC\_BLOCKF) can be used to encode Health Industry Barcode (HIBC) data which adds a leading '+' character and a modulo-49 check digit to the encoded data.

### 6.2.3 Code 16K (EN 12323)

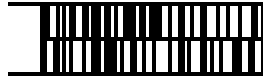


Figure 72: `zint -b CODE16K --compliantheight -d "ab0123456789"`

Code 16K uses a Code 128 based system which can stack up to 16 rows in a block. This gives a maximum data capacity of 77 characters or 154 numerical digits and includes two modulo-107 check digits. Code 16K also supports extended ASCII character encoding in the same manner as Code 128. GS1 data encoding is also supported. The minimum number of rows to use can be set using the `--rows` option (API `option_1`), with values from 2 to 16.

### 6.2.4 PDF417 (ISO 15438)



Figure 73: `zint -b PDF417 -d "PDF417"`

Heavily used in the parcel industry, the PDF417 symbology can encode a vast amount of data into a small space. Zint supports encoding up to the ISO standard maximum symbol size of 925 codewords which (at error correction level 0) allows a maximum data size of 1850 text characters, or 2710 digits.

The width of the generated PDF417 symbol can be specified at the command line using the `--cols` switch (API `option_2`) followed by a number between 1 and 30, the number of rows using the `--rows` switch (API `option_3`) followed by a number between 3 and 90, and the amount of error correction information can be specified by using the `--secure` switch (API `option_1`) followed by a number between 0 and 8 where the number of codewords used for error correction is determined by  $2^{(\text{value} + 1)}$ . The default level of error correction is determined by the amount of data being encoded.

This symbology uses Latin-1 character encoding by default but also supports the ECI encoding mechanism. A separate symbology ID (BARCODE\_HIBC\_PDF) can be used to encode Health Industry Barcode (HIBC) data.

PDF417 supports Structured Append of up to 99,999 symbols and an optional numeric ID of up to 30 digits, which can be set by using the `--structapp` option (see [4.16 Structured Append](#)) (API `structapp`). The ID consists of up to 10 triplets, each ranging from "000" to "899". For instance "123456789" would be a valid ID of 3 triplets. However "123456900" would not, as the last triplet "900" exceeds "899". The triplets are 0-filled, for instance "1234" becomes "123004". If an ID is not given, no ID is encoded.

### 6.2.5 Compact PDF417 (ISO 15438)



Figure 74: `zint -b PDF417COMP -d "PDF417"`

Previously known as Truncated PDF417, Compact PDF417 omits some per-row overhead to produce a narrower but less robust symbol. Options are the same as for PDF417 above.

### 6.2.6 MicroPDF417 (ISO 24728)



Figure 75: `zint -b MICROPDF417 -d "12345678"`

A variation of the PDF417 standard, MicroPDF417 is intended for applications where symbol size needs to be kept to a minimum. 34 predefined symbol sizes are available with 1 - 4 columns and 4 - 44 rows. The maximum size a MicroPDF417 symbol can hold is 250 alphanumeric characters or 366 digits. The amount of error correction used is dependent on symbol size. The number of columns used can be determined using the `--cols` switch (API `option_2`) as with PDF417.

This symbology uses Latin-1 character encoding by default but also supports the ECI encoding mechanism. A separate symbology ID (`BARCODE_HIBC_MICPDF`) can be used to encode Health Industry Barcode (HIBC) data. MicroPDF417 supports Structured Append the same as PDF417, for which see details.

## 6.2.7 GS1 DataBar Stacked (ISO 24724)

### 6.2.7.1 GS1 DataBar Stacked



Figure 76: `zint -b DBAR_STK --complantheight -d "9889876543210"`

A stacked variation of the GS1 DataBar Truncated symbol requiring the same input (see [6.1.11.1 GS1 DataBar Omnidirectional and GS1 DataBar Truncated](#)), this symbol is the same as the following DataBar Stacked Omnidirectional symbol except that its height is reduced, making it suitable for small items when omnidirectional scanning is not required. It can be generated with a two-dimensional component to make a composite symbol.

### 6.2.7.2 GS1 DataBar Stacked Omnidirectional



Figure 77: `zint -b DBAR_OMNSTK --complantheight -d "9889876543210"`

A stacked variation of the GS1 DataBar Omnidirectional symbol requiring the same input (see [6.1.11.1 GS1 DataBar Omnidirectional and GS1 DataBar Truncated](#)). The data is encoded in two rows of bars with a central finder pattern. This symbol can be generated with a two-dimensional component to make a composite symbol.

### 6.2.7.3 GS1 DataBar Expanded Stacked

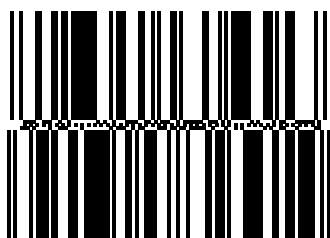


Figure 78: `zint -b DBAR_EXPSTK --complantheight -d "[01]98898765432106[3202]012345[15]991231"`

A stacked variation of the GS1 DataBar Expanded symbol for smaller packages. Input is the same as for GS1 DataBar Expanded (see [6.1.11.3 GS1 DataBar Expanded](#)). In addition the width of the symbol can be altered using the `--cols` switch (API `option_2`). In this case the number of columns (values 1 to 11) relates to the number of character pairs on each row of the symbol. Alternatively the `--rows` switch (API `option_3`) can be used to specify the maximum number of rows (values 2 to 11), and the number of columns will be adjusted accordingly. This symbol can be generated with a two-dimensional component to make a composite symbol. For symbols with a 2D component the number of columns must be at least 2.

### 6.2.8 Code 49

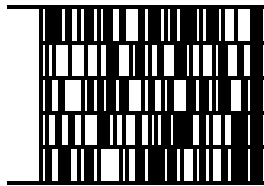


Figure 79: `zint -b CODE49 --compliantheight -d "MULTIPLE ROWS IN CODE 49"`

Developed in 1987 at Intermec, Code 49 is a cross between UPC and Code 39. It is one of the earliest stacked symbologies and influenced the design of Code 16K a few years later. It supports full 7-bit ASCII input up to a maximum of 49 characters or 81 numeric digits. GS1 data encoding is also supported. The minimum number of rows to use can be set using the `--rows` option (API `option_1`), with values from 2 to 8.



### 6.3 Composite Symbols (ISO 24723)

Composite symbols employ a mixture of components to give more comprehensive information about a product. The permissible contents of a composite symbol is determined by the terms of the GS1 General Specifications. Composite symbols consist of a linear component which can be an EAN, UPC, GS1-128 or GS1 DataBar symbol, a 2D component which is based on PDF417 or MicroPDF417, and a separator pattern. The type of linear component to be used is determined using the `-b` or `--barcode` switch (API `symbology`) as with other encoding methods. Valid values are shown below.

Table : Composite Symbology Values

Numeric Value	Name	Barcode Name
130	BARCODE_EANX_CC	Composite Symbol with EAN linear component
131	BARCODE_GS1_128_CC	Composite Symbol with GS1-128 linear component
132	BARCODE_DBAR_OMN_CC	Composite Symbol with GS1 DataBar Omnidirectional linear component
133	BARCODE_DBAR_LTD_CC	Composite Symbol with GS1 DataBar Limited linear component
134	BARCODE_DBAR_EXP_CC	Composite Symbol with GS1 DataBar Expanded linear component
135	BARCODE_UPCA_CC	Composite Symbol with UPC-A linear component
136	BARCODE_UPCE_CC	Composite Symbol with UPC-E linear component
137	BARCODE_DBAR_STK_CC	Composite Symbol with GS1 DataBar Stacked component
138	BARCODE_DBAR_OMNSTK_CC	Composite Symbol with GS1 DataBar Stacked Omnidirectional component
139	BARCODE_DBAR_EXPSTK_CC	Composite Symbol with GS1 DataBar Expanded Stacked component

The data to be encoded in the linear component of a composite symbol should be entered into a primary string with the data for the 2D component being entered in the normal way. To do this at the command prompt use the `--primary` switch (API `primary`). For example:

```
zint -b EANX_CC --mode=1 --primary=331234567890 -d "[99]1234-abcd"
```

This creates an EAN-13 linear component with the data "331234567890" and a 2D CC-A (see [below](#)) component with the data "(99)1234-abcd". The same results can be achieved using the API as shown below:

```
my_symbol->symbology = BARCODE_EANX_CC;
```

```
my_symbol->option_1 = 1;
```

```
strcpy(my_symbol->primary, "331234567890");
```

```
ZBarcode_Encode_and_Print(my_symbol, "[99]1234-abcd", 0, 0);
```

EAN-2 and EAN-5 add-on data can be used with EAN and UPC symbols using the `+` symbol as described in sections [6.1.3 Universal Product Code \(ISO 15420\)](#) and [6.1.4 European Article Number \(ISO 15420\)](#).

The 2D component of a composite symbol can use one of three systems: CC-A, CC-B and CC-C, as described below. The 2D component type can be selected automatically by Zint dependent on the length of the input string. Alternatively the three methods can be accessed using the `--mode` prompt (API `option_1`) followed by 1, 2 or 3 for CC-A, CC-B or CC-C respectively.

### 6.3.1 CC-A



Figure 80: `zint -b EANX_CC --compliantheight -d "[99]1234-abcd" --mode=1 --primary=331234567890`

This system uses a variation of MicroPDF417 which is optimised to fit into a small space. The size of the 2D component and the amount of error correction is determined by the amount of data to be encoded and the type of linear component which is being used. CC-A can encode up to 56 numeric digits or an alphanumeric string of shorter length. To select CC-A use `--mode=1` (API `option_1 = 1`).

### 6.3.2 CC-B



Figure 81: `zint -b EANX_CC --compliantheight -d "[99]1234-abcd" --mode=2 --primary=331234567890`

This system uses MicroPDF417 to encode the 2D component. The size of the 2D component and the amount of error correction is determined by the amount of data to be encoded and the type of linear component which is being used. CC-B can encode up to 338 numeric digits or an alphanumeric string of shorter length. To select CC-B use `--mode=2` (API `option_1 = 2`).

### 6.3.3 CC-C



Figure 82: `zint -b GS1_128_CC --compliantheight -d "[99]1234-abcd" --mode=3 --primary="[01]03312345678903"`

This system uses PDF417 and can only be used in conjunction with a GS1-128 linear component. CC-C can encode up to 2361 numeric digits or an alphanumeric string of shorter length. To select CC-C use `--mode=3` (API `option_1 = 3`).

## 6.4 Two-Track Symbols

### 6.4.1 Two-Track Pharmacode



Figure 83: `zint -b PHARMA_TWO --complianceheight -d "29876543"`

Developed by Laetus, Pharmacode Two-Track is an alternative system to Pharmacode One-Track (see [6.1.9 Pharmacode](#)) used for the identification of pharmaceuticals. The symbology is able to encode whole numbers between 4 and 64570080.

### 6.4.2 POSTNET



Figure 84: `zint -b POSTNET --complianceheight -d "12345678901"`

Used by the United States Postal Service until 2009, the POSTNET barcode was used for encoding zip-codes on mail items. POSTNET uses numerical input data and includes a modulo-10 check digit. While Zint will encode POSTNET symbols of up to 38 digits in length, standard lengths as used by USPS were *PostNet6* (5 digit ZIP input), *PostNet10* (5 digit ZIP + 4 digit user data) and *PostNet12* (5 digit ZIP + 6 digit user data).

### 6.4.3 PLANET



Figure 85: `zint -b PLANET --complianceheight -d "4012345235636"`

Used by the United States Postal Service until 2009, the PLANET (Postal Alpha Numeric Encoding Technique) barcode was used for encoding routing data on mail items. PLANET uses numerical input data and includes a modulo-10 check digit. While Zint will encode PLANET symbols of up to 38 digits in length, standard lengths used by USPS were *Planet12* (11 digit input) and *Planet14* (13 digit input).

## 6.5 4-State Postal Codes

### 6.5.1 Australia Post 4-State Symbols

#### 6.5.1.1 Customer Barcodes



Figure 86: `zint -b AUSPOST --compliantheight -d "96184209"`

Australia Post Standard Customer Barcode, Customer Barcode 2 and Customer Barcode 3 are 37-bar, 52-bar and 67-bar specifications respectively, developed by Australia Post for printing Delivery Point ID (DPID) and customer information on mail items. Valid data characters are 0-9, A-Z, a-z, space and hash (#). A Format Control Code (FCC) is added by Zint and should not be included in the input data. Reed-Solomon error correction data is generated by Zint. Encoding behaviour is determined by the length of the input data according to the formula shown in the following table:

Table : Australia Post Input Formats

Input Length	Required Input Format	Symbol Length	FCC	Encoding Table
8	99999999	37-bar	11	None
13	99999999AAAAA	52-bar	59	C
16	9999999999999999	52-bar	59	N
18	99999999AAAAAAAAA	67-bar	62	C
23	9999999999999999999999	67-bar	62	N

#### 6.5.1.2 Reply Paid Barcode



Figure 87: `zint -b AUSREPLY --compliantheight -d "12345678"`

A Reply Paid version of the Australia Post 4-State Barcode (FCC 45) which requires an 8-digit DPID input.

#### 6.5.1.3 Routing Barcode



Figure 88: `zint -b AUSROUTE --compliantheight -d "34567890"`

A Routing version of the Australia Post 4-State Barcode (FCC 87) which requires an 8-digit DPID input.

#### 6.5.1.4 Redirect Barcode



Figure 89: `zint -b AUSREDIRECT --compliantheight -d "98765432"`

A Redirection version of the Australia Post 4-State Barcode (FCC 92) which requires an 8-digit DPID input.

### 6.5.2 Dutch Post KIX Code



Figure 90: `zint -b KIX --compliantheight -d "2500GG30250"`

This symbology is used by Royal Dutch TPG Post (Netherlands) for Postal code and automatic mail sorting. Data input can consist of numbers 0-9 and letters A-Z and needs to be 11 characters in length. No check digit is included.

### 6.5.3 Royal Mail 4-State Customer Code (RM4SCC)



Figure 91: `zint -b RM4SCC --complianceheight -d "W1J0TR01"`

The RM4SCC standard is used by the Royal Mail in the UK to encode postcode and customer data on mail items. Data input can consist of numbers 0-9 and letters A-Z and usually includes delivery postcode followed by house number. For example "W1J0TR01" for 1 Piccadilly Circus in London. Check digit data is generated by Zint.

### 6.5.4 Royal Mail 4-State Mailmark



Figure 92: `zint -b MAILMARK --complianceheight -d "11000000000000XY11"`

Developed in 2014 as a replacement for RM4SCC this 4-state symbol includes Reed Solomon error correction. Input is a pre-formatted alphanumeric string of 22 (for Barcode C) or 26 (for Barcode L) characters, producing a symbol with 66 or 78 bars respectively. Some of the permitted inputs include a number of trailing space characters - these will be appended by Zint if not included in the input data.

### 6.5.5 USPS Intelligent Mail



Figure 93: `zint -b USPS_IMAIL --complianceheight -d "01234567094987654321-01234"`

Also known as the OneCode barcode and used in the US by the United States Postal Service (USPS), the Intelligent Mail system replaced the POSTNET and PLANET symbologies in 2009. Intelligent Mail is a fixed length (65-bar) symbol which combines routing and customer information in a single symbol. Input data consists of a 20 digit tracking code, followed by a dash (-), followed by a delivery point zip-code which can be 0, 5, 9 or 11 digits in length. For example all of the following inputs are valid data entries:

"01234567094987654321"

"01234567094987654321-01234"

"01234567094987654321-012345678"

"01234567094987654321-01234567891"

### 6.5.6 Japanese Postal Code



Figure 94: `zint -b JAPANPOST --complianceheight -d "15400233-16-4-205"`

Used for address data on mail items for Japan Post. Accepted values are 0-9, A-Z and dash (-). A modulo 19 check digit is added by Zint.

## 6.6 Matrix Symbols

### 6.6.1 Data Matrix (ISO 16022)



Figure 95: `zint -b HIBC_DM -d "/ACMRN123456/V200912190833" --fast --square`

Also known as Semacode this symbology was developed in 1989 by Acuity CiMatrix in partnership with the US DoD and NASA. The symbol can encode a large amount of data in a small area. Data Matrix encodes characters in the Latin-1 set by default but also supports encoding in other character sets using the ECI mechanism. It can also encode GS1 data. The size of the generated symbol can also be adjusted using the `--vers` option (API `option_2`) as shown in the table below. A separate symbology ID (`BARCODE_HIBC_DM`) can be used to encode Health Industry Barcode (HIBC) data. Note that only ECC200 encoding is supported, the older standards have now been removed from Zint.

Table : Data Matrix Sizes

Input	Symbol Size	Input	Symbol Size	Input	Symbol Size
1	10 x 10	11	36 x 36	21	104 x 104
2	12 x 12	12	40 x 40	22	120 x 120
3	14 x 14	13	44 x 44	23	132 x 132
4	16 x 16	14	48 x 48	24	144 x 144
5	18 x 18	15	52 x 52	25	8 x 18
6	20 x 20	16	64 x 64	26	8 x 32
7	22 x 22	17	72 x 72	28	12 x 26
8	24 x 24	18	80 x 80	28	12 x 36
9	26 x 26	19	88 x 88	29	16 x 36
10	32 x 32	20	96 x 96	30	16 x 48

When using automatic symbol sizes you can force Zint to use square symbols (versions 1-24) at the command line by using the option `--square` (API `option_3 = DM_SQUARE`).

Data Matrix Rectangular Extension (ISO/IEC 21471) codes may be generated with the following values as before:

Table : DMRE Sizes

Input	Symbol Size	Input	Symbol Size
31	8 x 48	40	20 x 36
32	8 x 64	41	20 x 44
33	8 x 80	42	20 x 64
34	8 x 96	43	22 x 48
35	8 x 120	44	24 x 48
36	8 x 144	45	24 x 64
37	12 x 64	46	26 x 40
38	12 x 88	47	26 x 48
39	16 x 64	48	26 x 64

DMRE symbol sizes may be activated in automatic size mode using the option `--dmre` (API `option_3 = DM_DMRE`).

GS1 data may be encoded using FNC1 (default) or GS as separator. Use the option `--gssep` to change to GS (API `output_options |= GS1_GS_SEPARATOR`).

For a faster but less optimal encoding, the `--fast` option (API `input_mode |= FAST_MODE`) may be used.

Data Matrix supports Structured Append of up to 16 symbols and a numeric ID (file identifications), which can be set by using the `--structapp` option (see [4.16 Structured Append](#)) (API `structapp`). The ID consists of 2 numbers ID1 and ID2, each of which can range from 1 to 254, and is specified as the single number  $ID1 * 1000 + ID2$ , so for instance ID1 "123" and ID2 "234" would be given as "123234". Note that both ID1 and ID2 must be non-zero, so e.g. "123000" or "000123" would be invalid IDs. If an ID is not given it defaults to "001001".

## 6.6.2 QR Code (ISO 18004)



Figure 96: `zint -b QRCODE -d "QR Code Symbol" --mask=5`

Also known as Quick Response Code this symbology was developed by Denso. Four levels of error correction are available using the `--secure` option (API `option_1`) as shown in the following table.

Table : QR Code ECC Levels

Input	ECC Level	Error Correction Capacity	Recovery Capacity
1	L	Approx 20% of symbol	Approx 7%
2	M	Approx 37% of symbol	Approx 15%
3	Q	Approx 55% of symbol	Approx 25%
4	H	Approx 65% of symbol	Approx 30%

The size of the symbol can be specified by setting the `--vers` option (API `option_2`) to the QR Code version required (1-40). The size of symbol generated is shown in the table below.

Table : QR Code Sizes

Input	Symbol Size	Input	Symbol Size	Input	Symbol Size
1	21 x 21	15	77 x 77	29	133 x 133
2	25 x 25	16	81 x 81	30	137 x 137
3	29 x 29	17	85 x 85	31	141 x 141
4	33 x 33	18	89 x 89	32	145 x 145
5	37 x 37	19	93 x 93	33	149 x 149
6	41 x 41	20	97 x 97	34	153 x 153
7	45 x 45	21	101 x 101	35	157 x 157
8	49 x 49	22	105 x 105	36	161 x 161
9	53 x 53	23	109 x 109	37	165 x 165
10	57 x 57	24	113 x 113	38	169 x 169
11	61 x 61	25	117 x 117	39	173 x 173
12	65 x 65	26	121 x 121	40	177 x 177
13	69 x 69	27	125 x 125		
14	73 x 73	28	129 x 129		

The maximum capacity of a QR Code symbol (version 40) is 7089 numeric digits, 4296 alphanumeric characters or 2953 bytes of data. QR Code symbols can also be used to encode GS1 data. QR Code symbols can by default encode either characters in the Latin-1 set or Kanji, Katakana and ASCII characters which are members of the Shift JIS encoding scheme. In addition QR Code supports other character sets using the ECI mechanism. Input should usually be entered as UTF-8 with conversion to Latin-1 or Shift JIS being carried out by Zint. A separate symbology ID (`BARCODE_HIBC_QR`) can be used to encode Health Industry Barcode (HIBC) data.

Non-ASCII data density may be maximized by using the `--fullmultibyte` switch (API `option_3` = `ZINT_FULL_MULTIBYTE`), but check that your barcode reader supports this before using.

QR Code has eight different masks designed to minimize unwanted patterns. The best mask to use is selected automatically by Zint but may be manually specified by using the `--mask` switch with values 0-7, or in the API by setting `option_3 = (N + 1) << 8` where N is 0-7. To use with `ZINT_FULL_MULTIBYTE` set

```
option_3 = ZINT_FULL_MULTIBYTE | (N + 1) << 8
```

QR Code supports Structured Append of up to 16 symbols and a numeric ID (parity), which can be set by using the `--structapp` option (see [4.16 Structured Append](#)) (API `structapp`). The parity ID ranges from 0 (default) to 255, and for full compliance should be set to the value obtained by XOR-ing together each byte of the complete data forming the sequence. Currently this calculation must be done outside of Zint.

### 6.6.3 Micro QR Code (ISO 18004)



Figure 97: `zint -b MICROQR -d "01234567"`

A miniature version of the QR Code symbol for short messages, Micro QR Code symbols can encode either Latin-1 characters or Shift JIS characters. Input should be entered as a UTF-8 stream with conversion to Latin-1 or Shift JIS being carried out automatically by Zint. A preferred symbol size can be selected by using the `--vers` option (API `option_2`), as shown in the table below. Note that versions M1 and M2 have restrictions on what characters can be encoded.

Table : Micro QR Code Sizes

Input	Version	Symbol Size	Allowed Characters
1	M1	11 x 11	Numeric only
2	M2	13 x 13	Numeric, uppercase letters, space, and the characters "\$%*+-./:"
3	M3	15 x 15	Latin-1 and Shift JIS
4	M4	17 x 17	Latin-1 and Shift JIS

Except for version M1, which is always ECC level L, the amount of ECC codewords can be adjusted using the `--secure` option (API `option_1`); however ECC level H is not available for any version, and ECC level Q is only available for version M4:

Input	ECC Level	Error Correction Capacity	Recovery Capacity	Available for Versions
1	L	Approx 20% of symbol	Approx 7%	M1, M2, M3, M4
2	M	Approx 37% of symbol	Approx 15%	M2, M3, M4
3	Q	Approx 55% of symbol	Approx 25%	M4

The defaults for symbol size and ECC level depend on the input and whether either of them is specified.

For barcode readers that support it, non-ASCII data density may be maximized by using the `--fullmultibyte` switch (API `option_3 = ZINT_FULL_MULTIBYTE`).

Micro QR Code has four different masks designed to minimize unwanted patterns. The best mask to use is selected automatically by Zint but may be manually specified by using the `--mask` switch with values 0-3, or in the API by setting `option_3 = (N + 1) << 8` where N is 0-3. To use with `ZINT_FULL_MULTIBYTE` set

```
option_3 = ZINT_FULL_MULTIBYTE | (N + 1) << 8
```



#### 6.6.4 Rectangular Micro QR Code (rMQR)



Figure 98: `zint -b RMQR -d "0123456"`

A rectangular version of QR Code, it is still under development, so it is recommended it should not yet be used for a production environment. Like QR Code, rMQR supports encoding of GS1 data, and either Latin-1 characters or Shift JIS characters, and other encodings using the ECI mechanism. As with other symbologies data should be entered as UTF-8 with conversion being handled by Zint. The amount of ECC codewords can be adjusted using the `--secure` option (`API option_1`), however only ECC levels M and H are valid for this type of symbol.

Table : rMQR ECC Levels

Input	ECC Level	Error Correction Capacity	Recovery Capacity
2	M	Approx 37% of symbol	Approx 15%
4	H	Approx 65% of symbol	Approx 30%

The preferred symbol sizes can be selected using the `--vers` option (`API option_2`) as shown in the table below. Input values between 33 and 38 fix the height of the symbol while allowing Zint to determine the minimum symbol width.

Table : rMQR Sizes

Input	Version	Symbol Size (HxW)	Input	Version	Symbol Size (HxW)
1	R7x43	7 x 73	20	R13x77	13 x 77
2	R7x59	7 x 59	21	R13x99	13 x 99
3	R7x77	7 x 77	22	R13x139	13 x 139
4	R7x99	7 x 99	23	R15x43	15 x 43
5	R7x139	7 x 139	24	R15x59	15 x 59
6	R9x43	9 x 43	25	R15x77	15 x 77
7	R9x59	9 x 59	26	R15x99	15 x 99
8	R9x77	9 x 77	27	R15x139	15 x 139
9	R9x99	9 x 99	28	R17x43	17 x 43
10	R9x139	9 x 139	29	R17x59	17 x 59
11	R11x27	11 x 27	30	R17x77	17 x 77
12	R11x43	11 x 43	31	R17x99	17 x 99
13	R11x59	11 x 59	32	R17x139	17 x 139
14	R11x77	11 x 77	33	R7xW	7 x automatic width
15	R11x99	11 x 99	34	R9xW	9 x automatic width
16	R11x139	11 x 139	35	R11xW	11 x automatic width
17	R13x27	13 x 27	36	R13xW	13 x automatic width
18	R13x43	13 x 43	37	R15xW	15 x automatic width
19	R13x59	13 x 59	38	R17xW	17 x automatic width

For barcode readers that support it, non-ASCII data density may be maximized by using the `--fullmultibyte` switch or in the API by setting `option_3 = ZINT_FULL_MULTIBYTE`.

### 6.6.5 UPNQR (Univerzalnega Plačilnega Naloga QR)



Figure 99: `zint -b UPNQR -i upn_utf8.txt --quietzones`

A variation of QR Code used by Združenje Bank Slovenije (Bank Association of Slovenia). The size, error correction level and ECI are set by Zint and do not need to be specified. UPNQR is unusual in that it uses Latin-2 (ISO/IEC 8859-2 plus ASCII) formatted data. Zint will accept UTF-8 data and convert it to Latin-2, or if your data is already Latin-2 formatted use the `--binary` switch (API `input_mode = DATA MODE`).

The following example creates a symbol from data saved as a Latin-2 file:

```
zint -o upnqr.png -b 143 --scale=3 --binary -i upn.txt
```

### 6.6.6 MaxiCode (ISO 16023)

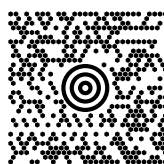


Figure 100: `zint -b MAXICODE -d "1Z00004951\GUPSN\G06X610\G159\G1234567\G1/1\G\GY\G1 MAIN ST\GNY\GNY\R\E" --esc --primary="152382802000000" --scmvv=96`

Developed by UPS the MaxiCode symbology employs a grid of hexagons surrounding a bulls-eye finder pattern. This symbology is designed for the identification of parcels. MaxiCode symbols can be encoded in one of five modes. In modes 2 and 3 MaxiCode symbols are composed of two parts named the primary and secondary messages. The primary message consists of a Structured Carrier Message which includes various data about the package being sent and the secondary message usually consists of address data in a data structure. The format of the primary message required by Zint is given in the following table:

Table : MaxiCode Structured Carrier Message Format

Characters	Meaning
1 - 9	Postcode data which can consist of up to 9 digits (for mode 2) or up to 6 alphanumeric characters (for mode 3). Remaining unused characters can be filled with the SPACE character (ASCII 32) or omitted (if omitted adjust the following character positions).
10 - 12	Three digit country code according to ISO 3166-1.
13 - 15	Three digit service code. This depends on your parcel courier.

The primary message can be set at the command prompt using the `--primary` switch (API `primary`). The secondary message uses the normal data entry method. For example:

```
zint -o test.eps -b 57 --primary="99999999840012" \
-d "Secondary Message Here"
```

When using the API the primary message must be placed in the `primary` string. The secondary is entered in the same way as described in [5.2 Encoding and Saving to File](#). When either of these modes is selected Zint will analyse the primary message and select either mode 2 or mode 3 as appropriate.

As a convenience the secondary message for modes 2 and 3 can be set to be prefixed by the ISO/IEC 15434 Format "01" (transportation) sequence "[>\R01\Gvv", where vv is a 2-digit version, by using the `--scmvv` switch (API `option_2 = vv + 1`). For example to use the common version "96" (ASC MH10/SC 8):

```
zint -b 57 --primary="152382802840001" --scmvv=96 --esc -d \
"1Z00004951\GUPSN\G06X610\G159\G1234567\G1/1\G\GY\G1 MAIN ST\GNY\GNY\R\E"
```

will prefix "[>\R01\G96" to the secondary message. (\R, \G and \E are the escape sequences for Record Separator, Group Separator and End of Transmission respectively - see [Table : Escape Sequences](#).)

Modes 4 to 6 can be accessed using the `--mode` switch (API `option_1`). Modes 4 to 6 do not have a primary message. For example:

```
zint -o test.eps -b 57 --mode=4 -d "A MaxiCode Message in Mode 4"
```

Mode 6 is reserved for the maintenance of scanner hardware and should not be used to encode user data.

This symbology uses Latin-1 character encoding by default but also supports the ECI encoding mechanism. The maximum length of text which can be placed in a MaxiCode symbol depends on the type of characters used in the text.

Example maximum data lengths are given in the table below:

Table : MaxiCode Data Length Maxima

Mode	Maximum Data Length for Capital Letters	Maximum Data Length for Numeric Digits	Number of Error Correction Codewords
2*	84	126	50
3*	84	126	50
4	93	138	50
5	77	113	66
6	93	138	50

\* - secondary only

MaxiCode supports Structured Append of up to 8 symbols, which can be set by using the `--structapp` option (see [4.16 Structured Append](#)) (API `structapp`). It does not support specifying an ID.

MaxiCode uses a different scaling than other symbols for raster output, see [4.9.2 MaxiCode Raster Scaling](#), and also for EMF vector output, when the scale is multiplied by 20 instead of 2.

### 6.6.7 Aztec Code (ISO 24778)



Figure 101: `zint -b AZTEC -d "123456789012"`

Invented by Andrew Longacre at Welch Allyn Inc in 1995 the Aztec Code symbol is a matrix symbol with a distinctive bulls-eye finder pattern. Zint can generate Compact Aztec Code (sometimes called Small Aztec Code) as well as 'full-range' Aztec Code symbols and by default will automatically select symbol type and size dependent on the length of the data to be encoded. Error correction codewords will normally be generated to fill at least 23% of the symbol. Two options are available to change this behaviour:

The size of the symbol can be specified using the `--vers` option (API `option_2`) to a value between 1 and 36 according to the following table. The symbols marked with an asterisk (\*) in the table below are 'compact'

symbols, meaning they have a smaller bulls-eye pattern at the centre of the symbol.

Table : Aztec Code Sizes

Input	Symbol Size	Input	Symbol Size	Input	Symbol Size
1	15 x 15*	13	53 x 53	25	105 x 105
2	19 x 19*	14	57 x 57	26	109 x 109
3	23 x 23*	15	61 x 61	27	113 x 113
4	27 x 27*	16	67 x 67	28	117 x 117
5	19 x 19	17	71 x 71	29	121 x 121
6	23 x 23	18	75 x 75	30	125 x 125
7	27 x 27	19	79 x 79	31	131 x 131
8	31 x 31	20	83 x 83	32	135 x 135
9	37 x 37	21	87 x 87	33	139 x 139
10	41 x 41	22	91 x 91	34	143 x 143
11	45 x 45	23	95 x 95	35	147 x 147
12	49 x 49	24	101 x 101	36	151 x 151

Note that in symbols which have a specified size the amount of error correction is dependent on the length of the data input and Zint will allow error correction capacities as low as 3 codewords.

Alternatively the amount of error correction data can be specified by setting the `--secure` option (API `option_1`) to a value from the following table:

Table : Aztec Code Error Correction Modes

Mode	Error Correction Capacity
1	>10% + 3 codewords
2	>23% + 3 codewords
3	>36% + 3 codewords
4	>50% + 3 codewords

It is not possible to select both symbol size and error correction capacity for the same symbol. If both options are selected then the error correction capacity selection will be ignored.

Aztec Code supports ECI encoding and can encode up to a maximum length of approximately 3823 numeric or 3067 alphabetic characters or 1914 bytes of data. A separate symbology ID (BARCODE\_HIBC\_AZTEC) can be used to encode Health Industry Barcode (HIBC) data.

Aztec Code supports Structured Append of up to 26 symbols and an optional alphanumeric ID of up to 32 characters, which can be set by using the `--structapp` option (see [4.16 Structured Append](#)) (API `structapp`). The ID cannot contain spaces. If an ID is not given, no ID is encoded.

#### 6.6.8 Aztec Runes (ISO 24778)



Figure 102: `zint -b AZRUNE -d "125"`

A truncated version of compact Aztec Code for encoding whole integers between 0 and 255, as defined in ISO/IEC 24778 Annex A. Includes Reed-Solomon error correction. It does not support Structured Append.

### 6.6.9 Code One



Figure 103: `zint -b CODEONE -d "1234567890123456789012"`

A matrix symbology developed by Ted Williams in 1992 which encodes data in a way similar to Data Matrix, Code One is able to encode the Latin-1 character set or GS1 data, and also supports the ECI mechanism. There are two types of Code One symbol - fixed-ratio symbols which are roughly square (versions A through to H) and variable-width versions (version S and T). These can be selected by using `--vers` (API `option_2`) as shown in the table below:

Table : Code One Sizes

Input	Version	Size (W x H)	Numeric Data Capacity	Alphanumeric Data Capacity
1	A	16 x 18	22	13
2	B	22 x 22	44	27
3	C	28 x 28	104	64
4	D	40 x 42	217	135
5	E	52 x 54	435	271
6	F	70 x 76	886	553
7	G	104 x 98	1755	1096
8	H	148 x 134	3550	2218
9	S	width x 8	18	N/A
10	T	width x 16	90	55

Version S symbols can only encode numeric data. The width of version S and version T symbols is determined by the length of the input data.

Code One supports Structured Append of up to 128 symbols, which can be set by using the `--structapp` option (see [4.16 Structured Append](#)) (API `structapp`). It does not support specifying an ID. Structured Append is not supported with GS1 data nor for Version S symbols.

### 6.6.10 Grid Matrix

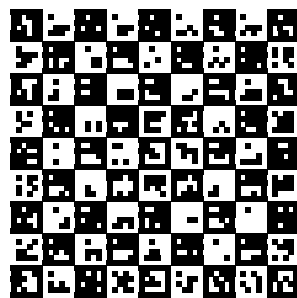


Figure 104: `zint -b GRIDMATRIX --eci=29 -d "AAT2556 电池充电器 + 降压转换器 200mA 至 2A tel:86 019 82512738"`

Grid Matrix groups modules in a chequerboard pattern, and by default supports the GB 2312 standard set, which includes Hanzi, ASCII and a small number of ISO/IEC 8859-1 characters. Input should be entered as UTF-8 with conversion to GB 2312 being carried out automatically by Zint. The symbology also supports the ECI mechanism. Support for GS1 data has not yet been implemented.

The size of the symbol and the error correction capacity can be specified. If you specify both of these values then Zint will make a 'best-fit' attempt to satisfy both conditions. The symbol size can be specified using the `--vers` option (API `option_2`), and the error correction capacity can be specified by using the `--secure` option (API `option_1`), according to the following tables:

Table : Grid Matrix Sizes

Input	Symbol Size	Input	Symbol Size
1	18 x 18	8	102 x 102
2	30 x 30	9	114 x 114
3	42 x 42	10	126 x 126
4	54 x 54	11	138 x 138
5	66 x 66	12	150 x 150
6	78 x 78	13	162 x 162
7	90 x 90		

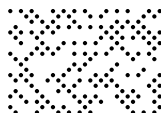
Table : Grid Matrix Error Correction Modes

Mode	Error Correction Capacity
1	Approximately 10%
2	Approximately 20%
3	Approximately 30%
4	Approximately 40%
5	Approximately 50%

Non-ASCII data density may be maximized by using the `--fullmultibyte` switch (API `option_3 = ZINT_FULL_MULTIBYTE`), but check that your barcode reader supports this before using.

Grid Matrix supports Structured Append of up to 16 symbols and a numeric ID (file signature), which can be set by using the `--structapp` option (see [4.16 Structured Append](#)) (API `structapp`). The ID ranges from 0 (default) to 255.

### 6.6.11 DotCode

Figure 105: `zint -b DOTCODE -d "[01]00012345678905[17]201231[10]ABC123456" --gs1`

DotCode uses a grid of dots in a rectangular formation to encode characters up to a maximum of approximately 450 characters (or 900 numeric digits). The symbology supports ECI encoding and GS1 data encoding. By default Zint will produce a symbol which is approximately square, however the width of the symbol can be adjusted by using the `--cols` option (API `option_2`) (maximum 200). Outputting DotCode to raster images (BMP, GIF, PCX, PNG, TIF) will require setting the scale of the image to a larger value than the default (e.g. approximately 10) for the dots to be plotted correctly. Approximately 33% of the resulting symbol is comprised of error correction codewords.

DotCode has two sets of 4 masks, designated 0-3 and 0'-3', the second "prime" set being the same as the first with corners lit. The best mask to use is selected automatically by Zint but may be manually specified by using the `--mask` switch with values 0-7, where 4-7 denote 0'-3', or in the API by setting `option_3 = (N + 1) << 8` where N is 0-7.

DotCode supports Structured Append of up to 35 symbols, which can be set by using the `--structapp` option (see [4.16 Structured Append](#)) (API `structapp`). It does not support specifying an ID.

## 6.6.12 Han Xin Code (ISO 20830)

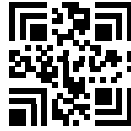


Figure 106: `zint -b HANXIN -d "Hanxin Code symbol"`

Also known as Chinese Sensible Code, Han Xin is capable of encoding characters in either the Latin-1 character set or the GB 18030 character set (which is a UTF, i.e. includes all Unicode characters, optimized for Chinese characters) and is also able to support the ECI mechanism. Support for the encoding of GS1 data has not yet been implemented.

The size of the symbol can be specified using the `--vers` option (API `option_2`) to a value between 1 and 84 according to the following table.

Table : Han Xin Sizes

Input	Symbol Size	Input	Symbol Size	Input	Symbol Size
1	23 x 23	29	79 x 79	57	135 x 135
2	25 x 25	30	81 x 81	58	137 x 137
3	27 x 27	31	83 x 83	59	139 x 139
4	29 x 29	32	85 x 85	60	141 x 141
5	31 x 31	33	87 x 87	61	143 x 143
6	33 x 33	34	89 x 89	62	145 x 145
7	35 x 35	35	91 x 91	63	147 x 147
8	37 x 37	36	93 x 93	64	149 x 149
9	39 x 39	37	95 x 95	65	151 x 151
10	41 x 41	38	97 x 97	66	153 x 153
11	43 x 43	39	99 x 99	67	155 x 155
12	45 x 45	40	101 x 101	68	157 x 157
13	47 x 47	41	103 x 103	69	159 x 159
14	49 x 49	42	105 x 105	70	161 x 161
15	51 x 51	43	107 x 107	71	163 x 163
16	53 x 53	44	109 x 109	72	165 x 165
17	55 x 55	45	111 x 111	73	167 x 167
18	57 x 57	46	113 x 113	74	169 x 169
19	59 x 59	47	115 x 115	75	171 x 171
20	61 x 61	48	117 x 117	76	173 x 173
21	63 x 63	49	119 x 119	77	175 x 175
22	65 x 65	50	121 x 121	78	177 x 177
23	67 x 67	51	123 x 123	79	179 x 179
24	69 x 69	52	125 x 125	80	181 x 181
25	71 x 71	53	127 x 127	81	183 x 183
26	73 x 73	54	129 x 129	82	185 x 185
27	75 x 75	55	131 x 131	83	187 x 187
28	77 x 77	56	133 x 133	84	189 x 189

There are four levels of error correction capacity available for Han Xin Code which can be set by using the `--secure` option (API `option_1`) to a value from the following table:

Table : Han Xin Error Correction Modes

Mode	Recovery Capacity
1	Approx 8%
2	Approx 15%
3	Approx 23%
4	Approx 30%

Non-ASCII data density may be maximized by using the `--fullmultibyte` switch (API `option_3 = ZINT_FULL_MULTIBYTE`), but check that your barcode reader supports this before using.

Han Xin has four different masks designed to minimize unwanted patterns. The best mask to use is selected automatically by Zint but may be manually specified by using the `--mask` switch with values 0-3, or in the API by setting `option_3 = (N + 1) << 8` where N is 0-3. To use with `ZINT_FULL_MULTIBYTE` set

```
option_3 = ZINT_FULL_MULTIBYTE | (N + 1) << 8
```

### 6.6.13 Ultracode

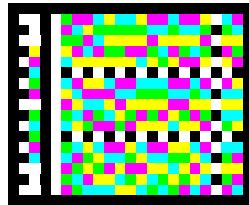


Figure 107: `zint -b ULTRA -d "HEIMASÍÐA KENNARAHÁSKÓLA ÍSLANDS"`

This symbology uses a grid of coloured elements to encode data. ECI and GS1 modes are supported. The amount of error correction can be set using the `--secure` option (API `option_1`) to a value as shown in the following table:

Table : Ultracode Error Correction Values

Value	EC Level	Amount of symbol holding error correction data
1	EC0	0% - Error detection only
2	EC1	Approx 5%
3	EC2	Approx 9% - Default value
4	EC3	Approx 17%
5	EC4	Approx 25%
6	EC5	Approx 33%

Zint does not currently implement data compression by default, but this can be initiated through the API by setting

```
symbol->option_3 = ULTRA_COMPRESSION;
```

WARNING: Ultracode data compression is experimental and should not be used in a production environment.

Revision 2 of Ultracode (2021) which swops and inverts the DCCU and DCCL tiles may be specified using `--vers=2` (API `option_2 = 2`).

Ultracode supports Structured Append of up to 8 symbols and an optional numeric ID (File Number), which can be set by using the `--structapp` option (see [4.16 Structured Append](#)) (API `structapp`). The ID ranges from 1 to 80088. If an ID is not given, no ID is encoded.



## 6.7 Other Barcode-Like Markings

### 6.7.1 Facing Identification Mark (FIM)

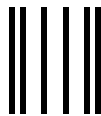


Figure 108: `zint -b FIM --compliantheight -d "C"`

Used by the United States Postal Service (USPS), the FIM symbology is used to assist automated mail processing. There are only 5 valid symbols which can be generated using the characters A-E as shown in the table below.

Table : Valid FIM Characters

Code Letter	Usage
A	Used for courtesy reply mail and metered reply mail with a pre-printed POSTNET symbol.
B	Used for business reply mail without a pre-printed zip code.
C	Used for business reply mail with a pre-printed zip code.
D	Used for Information Based Indicia (IBI) postage.
E	Used for customized mail with a USPS Intelligent Mail barcode.

### 6.7.2 Flattermarken

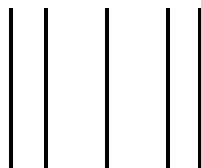


Figure 109: `zint -b FLAT -d "1304056"`

Used for the recognition of page sequences in print-shops, the Flattermarken is not a true barcode symbol and requires precise knowledge of the position of the mark on the page. The Flattermarken system can encode numeric data up to a maximum of 90 digits and does not include a check digit.

### 6.7.3 DAFT Code



Figure 110: `zint -b DAFT -d "AAFDTTDAFADTFTTFFFDATFTADTTFFTDFAFDTF" --height=8.494 --vers=256`

This is a method for creating 4-state codes where the data encoding is provided by an external program. Input data should consist of the letters 'D', 'A', 'F' and 'T' where these refer to descender, ascender, full (ascender and descender) and tracker (neither ascender nor descender) respectively. All other characters are invalid. The ratio of the tracker size to full height can be given in thousandths (permille) using the `--vers` option (API option\_2). The default value is 250 (25%).

For example the following

```
zint -b DAFT -d AAFDTTDAFADTFTTFFFDATFTADTTFFTDFAFDTF --height=8.494 --vers=256
```

produces the same barcode (see [6.5.3 Royal Mail 4-State Customer Code \(RM4SCC\)](#)) as

```
zint -b RM4SCC --compliantheight -d "W1J0TR01"
```

## 7. Legal and Version Information

### 7.1 License

Zint, libzint and Zint Barcode Studio are Copyright © 2022 Robin Stuart. All historical versions are distributed under the GNU General Public License version 3 or later. Version 2.5 (and later) is released under a dual license: the encoding library is released under the BSD license whereas the GUI, Zint Barcode Studio, is released under the GNU General Public License version 3 or later.

Telepen is a trademark of SB Electronic Systems Ltd.

QR Code is a registered trademark of Denso Wave Incorporated.

Microsoft, Windows and the Windows logo are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Mac and macOS are trademarks of Apple Inc., registered in the U.S. and other countries.

Zint.org.uk website design and hosting provided by Robert Elliott.

### 7.2 Patent Issues

All of the code in Zint is developed using information in the public domain, usually freely available on the Internet. Some of the techniques used may be subject to patents and other intellectual property legislation. It is my belief that any patents involved in the technology underlying symbologies utilised by Zint are 'unadopted', that is the holder does not object to their methods being used.

Any methods patented or owned by third parties or trademarks or registered trademarks used within Zint or in this document are and remain the property of their respective owners and do not indicate endorsement or affiliation with those owners, companies or organisations.

### 7.3 Version Information

The current stable version of Zint is 2.10.0, released on 14th August 2021.

See "ChangeLog" in the project root directory for information on all releases.

### 7.4 Sources of Information

Below is a list of some of the sources used in rough chronological order:

- Nick Johnson's Barcode Specifications
- Bar Code 1 Specification Source Page
- SB Electronic Systems Telepen website
- Pharmacode specifications from Laetus
- Morovia RM4SCC specification
- Australia Post's 'A Guide to Printing the 4-State Barcode' and bcsample source code
- Plessey algorithm from GNU-Barcode v0.98 by Leonid A. Broukhis
- GS1 General Specifications v 8.0 Issue 2
- PNG: The Definitive Guide and wpng source code by Greg Reolofs
- PDF417 specification and pdf417 source code by Grand Zebu
- Barcode Reference, TBarCode/X User Documentation and TBarCode/X demonstration program from Tec-It
- IEC16022 source code by Stefan Schmidt et al
- United States Postal Service Specification USPS-B-3200
- Adobe Systems Incorporated Encapsulated PostScript File Format Specification
- BSI Online Library
- Libdmtx Data Matrix ECC200 decoding library

### 7.5 Standards Compliance

Zint was developed to provide compliance with the following British and international standards:

- BS EN 798:1996 Bar coding - Symbology specifications - 'Codabar'

- BS EN 12323:2005 AIDC technologies - Symbology specifications - Code 16K
- ISO/IEC 15420:2009 Information technology - Automatic identification and data capture techniques - EAN/UPC bar code symbology specification
- ISO/IEC 15417:2007 Information technology - Automatic identification and data capture techniques - Code 128 bar code symbology specification
- ISO/IEC 15438:2015 Information technology - Automatic identification and data capture techniques - PDF417 bar code symbology specification
- ISO/IEC 16022:2006 Information technology - Automatic identification and data capture techniques - Data Matrix ECC200 bar code symbology specification
- ISO/IEC 16023:2000 Information technology - International symbology specification - MaxiCode
- ISO/IEC 16388:2007 Information technology - Automatic identification and data capture techniques - Code 39 bar code symbology specification
- ISO/IEC 18004:2015 Information technology - Automatic identification and data capture techniques - QR Code bar code symbology specification
- ISO/IEC 20830:2021 Information technology - Automatic identification and data capture techniques - Han Xin Code bar code symbology specification
- ISO/IEC 24723:2010 Information technology - Automatic identification and data capture techniques - GS1 Composite bar code symbology specification
- ISO/IEC 24724:2011 Information technology - Automatic identification and data capture techniques - GS1 DataBar bar code symbology specification
- ISO/IEC 24728:2006 Information technology - Automatic identification and data capture techniques - MicroPDF417 bar code symbology specification
- ISO/IEC 24778:2008 Information technology - Automatic identification and data capture techniques - Aztec Code bar code symbology specification
- ISO/IEC JTC1/SC31N000 (Draft 2019-6-24) Information technology - Automatic identification and data capture techniques - Rectangular Micro QR Code (rMQR) bar code symbology specification
- ISO/IEC 16390:2007 Information technology - Automatic identification and data capture techniques - Interleaved 2 of 5 bar code symbology specification
- ISO/IEC 21471:2020 Information technology - Automatic identification and data capture techniques - Extended rectangular data matrix (DMRE) bar code symbology specification
- Uniform Symbology Specification Code One (AIM Inc., 1994)
- ANSI/AIM BC12-1998 - Uniform Symbology Specification Channel Code
- ANSI/AIM BC6-2000 - Uniform Symbology Specification Code 49
- ANSI/AIM BC5-1995 - Uniform Symbology Specification Code 93
- ANSI/HIBC 2.6-2016 - The Health Industry Bar Code (HIBC) Supplier Labeling Standard
- AIM ISS-X-24 - Uniform Symbology Specification Codablock-F
- AIM TSC1705001 (v 4.0 Draft 0.15) - Information technology - Automatic identification and data capture techniques - Bar code symbology specification - DotCode (Revised 28th May 2019)
- AIMD014 (v 1.63) - Information technology, Automatic identification and data capture techniques - Bar code symbology specification - Grid Matrix (Released 9th Dec 2008)
- AIMD/TSC15032-43 (v 0.99c) - International Technical Specification - Ultracode Symbology (Draft) (Released 4th Nov 2015)
- GS1 General Specifications Release 22.0 (Jan 2022)
- AIM ITS/04-001 International Technical Standard - Extended Channel Interpretations Part 1: Identification Schemes and Protocol (Released 24th May 2004)
- AIM ITS/04-023 International Technical Standard - Extended Channel Interpretations Part 3: Register (Version 2, February 2022)

## A. Character Encoding

This section is intended as a quick reference to the character sets used by Zint. All symbologies use standard ASCII input as shown in section A.1, but some support extended characters as shown in the subsequent [A.2 Latin Alphabet No. 1 \(ISO/IEC 8859-1\)](#).

### A.1 ASCII Standard

The ubiquitous ASCII standard is well known to most computer users. It's reproduced here for reference.

Table : ASCII

Hex	0	1	2	3	4	5	6	7
0	NUL	DLE	SPACE	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	TAB	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

### A.2 Latin Alphabet No. 1 (ISO/IEC 8859-1)

ISO/IEC 8859-1 defines additional characters common in western European languages like French, German, Italian and Spanish. This extension is the default encoding of many barcodes (see [Table : Default Character Sets](#)) when a codepoint above hex 9F is encoded. Note that codepoints hex 80 to 9F are not defined.

Table : ISO/IEC 8859-1

Hex	8	9	A	B	C	D	E	F
0			NBSP	°	À	Ð	à	ð
1			í	±	Á	Ñ	á	ñ
2			¢	²	Â	Ò	â	ò
3			£	³	Ã	Ó	ã	ó
4			¤	´	Ä	Ô	ä	ô
5			¥	µ	Å	Õ	å	õ
6			¦	¶	Æ	Ö	æ	ö
7			§	·	Ç	×	ç	÷
8			¨	¸	È	Ø	è	ø
9			©	¹	É	Ù	é	ù
A			ª	º	Ê	Ú	ê	ú
B			«	»	Ë	Û	ë	û
C			¬	¼	Ì	Ü	ì	ü
D			SHY	½	Í	Ý	í	ý
E			®	¾	Î	Þ	î	þ
F			—	¿	Ï	ß	ï	ÿ

## B. CLI Help

Zint version 2.10.0.9

Encode input data in a barcode and save as BMP/EMF/EPS/GIF/PCX/PNG/SVG/TIF/TXT

<code>-b, --barcode=TYPE</code>	Number or name of barcode type. Default is 20 (CODE128)
<code>--addongap=NUMBER</code>	Set add-on gap in multiples of X-dimension for UPC/EAN
<code>--batch</code>	Treat each line of input file as a separate data set
<code>--bg=COLOUR</code>	Specify a background colour (in hex RGB/RGBA)
<code>--binary</code>	Treat input as raw binary data
<code>--bind</code>	Add boundary bars
<code>--bold</code>	Use bold text
<code>--border=NUMBER</code>	Set width of border in multiples of X-dimension
<code>--box</code>	Add a box around the symbol
<code>--cmyk</code>	Use CMYK colour space in EPS/TIF symbols
<code>--cols=NUMBER</code>	Set the number of data columns in symbol
<code>--compliantheight</code>	Warn if height not compliant, and use standard default
<code>-d, --data=DATA</code>	Set the symbol data content (segment 0)
<code>--direct</code>	Send output to stdout
<code>--dmre</code>	Allow Data Matrix Rectangular Extended
<code>--dotsize=NUMBER</code>	Set radius of dots in dotty mode
<code>--dotty</code>	Use dots instead of squares for matrix symbols
<code>--dump</code>	Dump hexadecimal representation to stdout
<code>-e, --ecinos</code>	Display ECI (Extended Channel Interpretation) table
<code>--eci=NUMBER</code>	Set the ECI code for the data (segment 0)
<code>--esc</code>	Process escape characters in input data
<code>--fast</code>	Use faster encodation (Data Matrix)
<code>--fg=COLOUR</code>	Specify a foreground colour (in hex RGB/RGBA)
<code>--filetype=TYPE</code>	Set output file type BMP/EMF/EPS/GIF/PCX/PNG/SVG/TIF/TXT
<code>--fullmultibyte</code>	Use multibyte for binary/Latin (QR/Han Xin/Grid Matrix)
<code>--gs1</code>	Treat input as GS1 compatible data
<code>--gs1nocheck</code>	Do not check validity of GS1 data
<code>--gs1parens</code>	Process parentheses "()" as GS1 AI delimiters, not "[]"
<code>--gssep</code>	Use separator GS for GS1 (Data Matrix)
<code>--guarddescent=NUMBER</code>	Set height of guard bar descent in X-dims (UPC/EAN)
<code>-h, --help</code>	Display help message
<code>--height=NUMBER</code>	Set height of symbol in multiples of X-dimension
<code>--heightperrow</code>	Treat height as per-row
<code>-i, --input=FILE</code>	Read input data from FILE
<code>--init</code>	Create reader initialisation/programming symbol
<code>--mask=NUMBER</code>	Set masking pattern to use (QR/Han Xin/DotCode)
<code>--mirror</code>	Use batch data to determine filename
<code>--mode=NUMBER</code>	Set encoding mode (MaxiCode/Composite)
<code>--nobackground</code>	Remove background (EMF/EPS/GIF/PNG/SVG/TIF only)
<code>--noquietzones</code>	Disable default quiet zones
<code>--notext</code>	Remove human readable text
<code>-o, --output=FILE</code>	Send output to FILE. Default is out.png
<code>--primary=STRING</code>	Set primary message (MaxiCode/Composite)
<code>--quietzones</code>	Add compliant quiet zones
<code>-r, --reverse</code>	Reverse colours (white on black)
<code>--rotate=NUMBER</code>	Rotate symbol by NUMBER degrees
<code>--rows=NUMBER</code>	Set number of rows (Codablock-F/PDF417)
<code>--scale=NUMBER</code>	Adjust size of X-dimension
<code>--scmvv=NUMBER</code>	Prefix SCM with "[>\R01\Gvv" (vv is NUMBER) (MaxiCode)
<code>--secure=NUMBER</code>	Set error correction level (ECC)
<code>--segN=ECI,DATA</code>	Set the ECI & data content for segment N, where N 1 to 9
<code>--separator=NUMBER</code>	Set height of row separator bars (stacked symbologies)
<code>--small</code>	Use small text
<code>--square</code>	Force Data Matrix symbols to be square
<code>--structapp=I,C[,ID]</code>	Set Structured Append info (I index, C count)
<code>-t, --types</code>	Display table of barcode types

<code>--vers=NUMBER</code>	Set symbol version (size, check digits, other options)
<code>-v, --version</code>	Display Zint version
<code>--vwhitesp=NUMBER</code>	Set height of vertical whitespace in multiples of X-dim
<code>-w, --whitesp=NUMBER</code>	Set width of horizontal whitespace in multiples of X-dim
<code>--werror</code>	Convert all warnings into errors