Zint Barcode Generator and Zint Barcode Studio User Manual

This version of the manual relates to Zint version 2.4.2.

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 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 Unicode (UTF-8) or a raw 8-bit data stream. The image can be rendered as either a Portable Network Graphic (PNG) image, as Encapsulated Post Script (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, in the case of PNG images, the orientation of the image.

If you find this project useful then please consider making a donation. Your support will ensure that we are able to continue to purchase and implement new barcode standards as they become available from various standards organisations. To make a donation go to:

http://sourceforge.net/project/project_donations.php?group_id=199350

Getting Help

The pages on this site should help you to get the most out of Zint. If, however, you have specific requirements or questions or wish to report a bug then either join the mailing list at

https://lists.sourceforge.net/lists/listinfo/zint-barcode

or send an e-mail to

zint-barcode@lists.sourceforge.net

Before posting to this list please note the following points...

- You do not need to join the list to post messages, although joining the list will usually mean your request is answered more promptly.
- Zint is primarily developed for Linux. If you are using another platform then we will be less

able to help you, although we will do so if we can. We cannot provide support for commercial packages such as MS Office or Crystal Reports.

- Always ensure you are using the latest version of Zint before posting bug reports - the bug you are reporting may have been fixed already.
- Please remember to state what operating system you are using and include enough information to allow us to reproduce the error - including input data if appropriate.
- Please DO NOT post messages asking for us to change the license arrangements. You will be ignored. If you want a barcode encoder with a different license then please look elsewhere (see below).
- Please remember that Zint is developed by volunteers - don't be surprised if we are unable to help you or if it takes a long time to answer your questions.

What's In a Name?

In best GNU fashion the name "Zint" is a recursive acronym for "Zint is not Tec-It". Tec-It is an Austrian ISV who specialise in AIDC technologies and in particular their commercial products "Barcode Studio" and "TBarcode/X" perform very similar functions to Zint. In fact the design of Zint is heavily influenced by these products with the hope that it can provide a close to 'drop-in replacement' capability.

Please note, however, that Zint is released under the GNU General Public License. This means that, for developers, it is only suitable for use with other open source packages. Zint cannot be released under any other license. If you see a copy of Zint released under any other license then please report it. If you require commercially licensed software to create barcodes then please contact Tec-It.

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 first. Note that you will need both libpng and libpng-devel packages. If you want to take advantage of Zint Barcode Studio you will also need the Qt libraries pre-installed.

Once you have fulfilled these requirements unzip the source code tarball and follow these steps in the top directory:

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

The command line program can be accessed by typing

```
zint {options} -d {data}
```

Notice that the data needs to be entered after all other options. Any options given after the data will be ignored. The GUI can be accessed by typing

```
zint-qt
```

To test that the installation has been sucessful a shell script is included in the /frontend folder. To run the test type

```
./test.sh
```

This should create numerous files showing the many modes of operation which are available from Zint.

2.2 Microsoft Windows

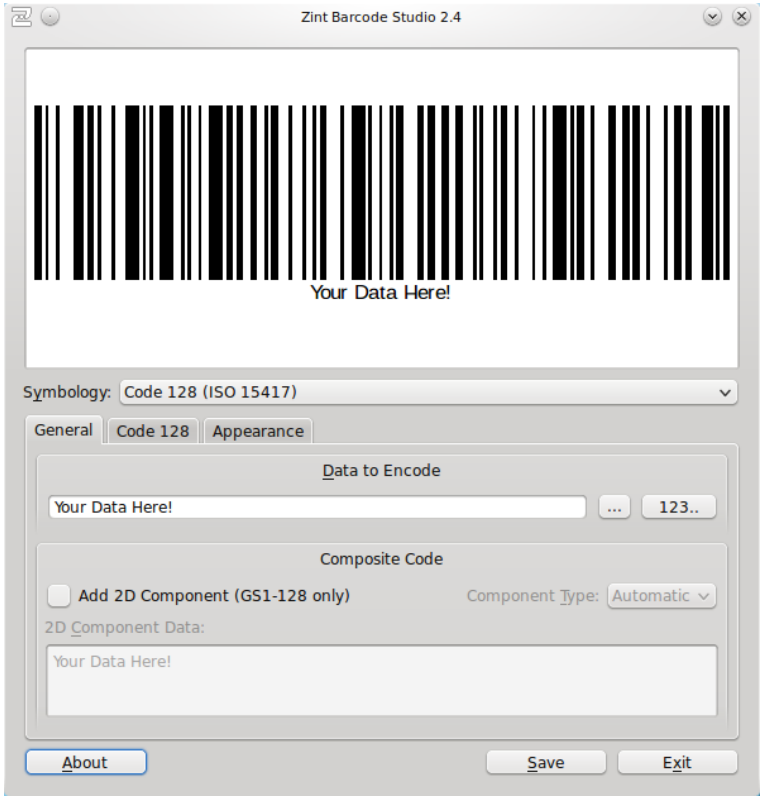
To run Zint Barcode Studio on Windows simply download and run the installation executable and follow the instructions on-screen.

2.3 Apple Mac OSX

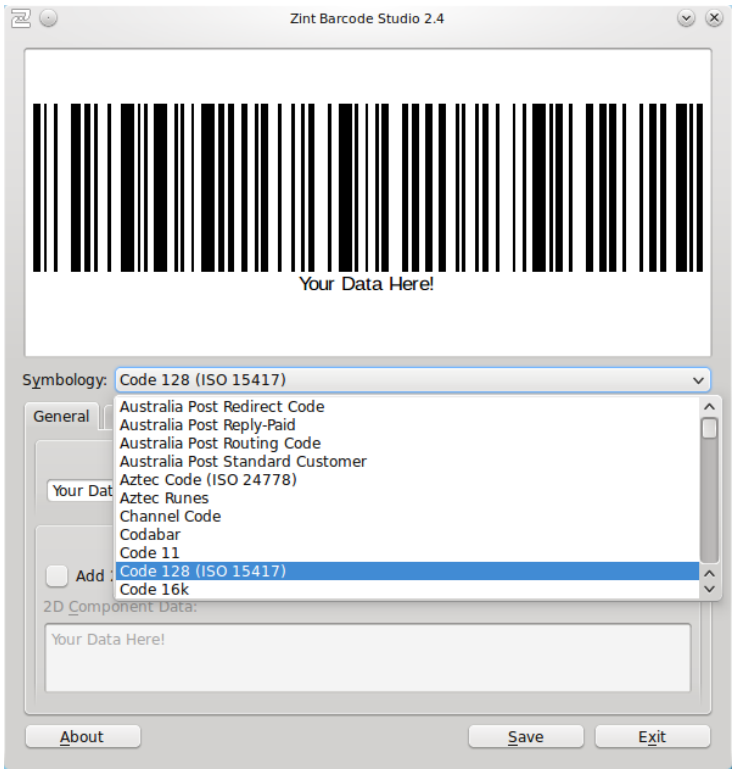
Zint can be compiled on OSX from the command line using the same steps as shown for Linux above. Currently the Zint Barcode Studio GUI is not known to work on OSX.

3. Using Zint Barcode Studio

Below is a brief guide to Zint Barcode Studio which is the graphical user interface for the Zint package.

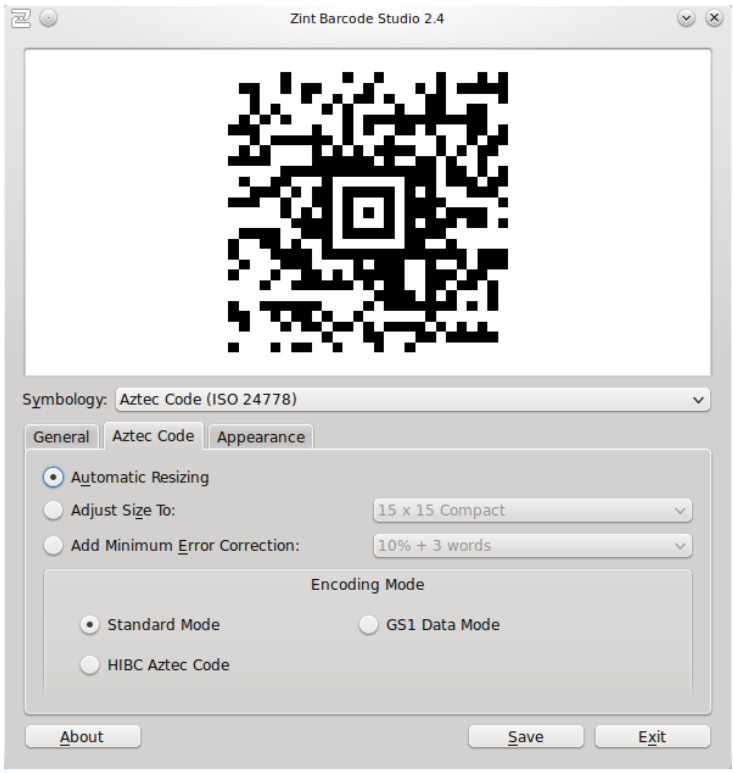


This is the main window of Zint Barcode Studio. The top of the window shows a preview of the barcode which the current settings would create. These settings can be changed using the controls below. The top most text box on this first tab allows you to enter the data to be encoded. When you are happy with your settings you can use the Save button to save the resulting image to file as a PNG, EPS or SVG image. The two sliders next to the preview allow you to change the orientation and scale of the preview image but do not affect the saved image.



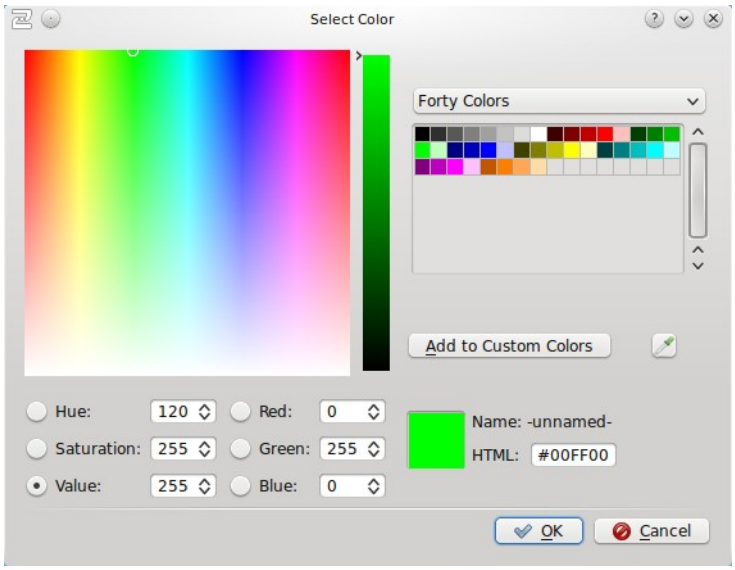
The Symbology drop-down box gives access to all of the symbologies supported by Zint shown in alphabetical order.

At the bottom of the screen 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 AI information from data as shown here.

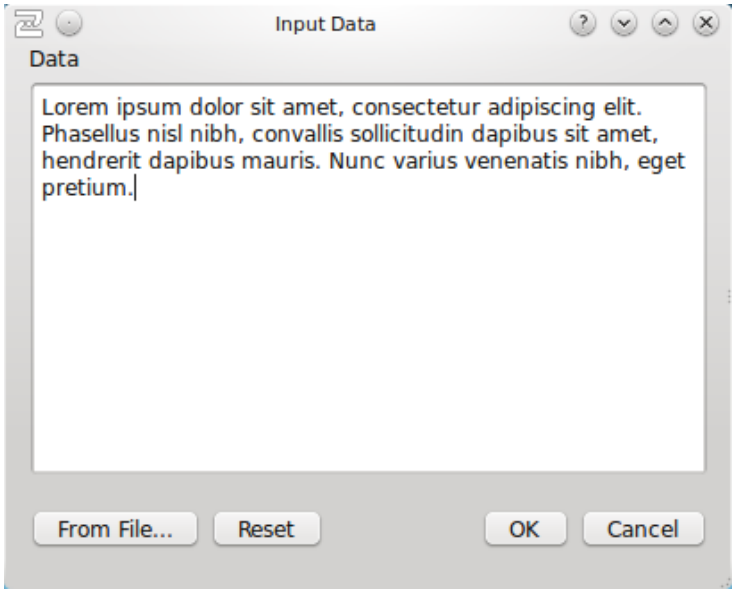


With some symbologies extra options area available to fine-tune the appearance or the content of the symbol generated. These are given in a second tab. Here the method is shown for adjusting the size of an Aztec Code symbol.

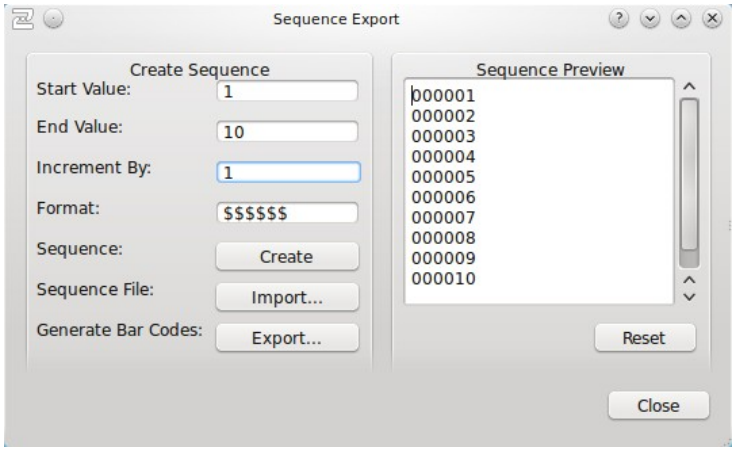
The appearance tab can be used to adjust the dimensions and other properties of the symbol. The height value affects the height of 1-dimensional symbologies. Boundary bars can be added and adjusted and the size of the saved image can be determined.



A colour dialog is used to adjust the colour of the foreground and background of the generated image. Click on “Foreground Colour” or “Background Colour” respectively.

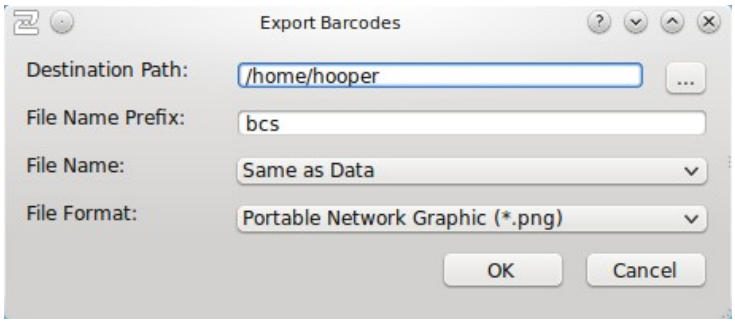


Clicking on the ellipsis next to the "Data to Encode" text box opens a larger window which can be used to enter longer strings of text. You can also use this window to get data from file.



Clicking on the sequence button (labelled "123..") 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 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:

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



The Export Dialog sets the parameters for exporting a sequence of barcode images. Here you can set the file name and the output image format. Note that the symbology, colour and other formatting information is taken from the main window.

4. Using the Command Line

This page describes how to encode data using the command line front end program.

4.1 Inputting data

The data to encode can be entered at the command line using the -d 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. The -d switch and the input data should always be the last entry on the command line input. Any options given after -d will be ignored.

The data input to Zint is assumed to be encoded in Unicode (UTF-8) format. If you are encoding characters beyond the 7-bit ASCII set on a platform which does not use Unicode by default then some corruption of the data may occur.

Non-printing characters can be entered on the command line using the backslash (\) as an escape character. Permissable characters are shown in the table below. Note that only applies on the command line.

Escape Character	ASCII Equivalent	Interpretation
\0	0x00	Null
\E	0x04	End of Transmission
\a	0x07	Bell
\b	0x08	Backspace
\t	0x09	Horizontal Tab
\n	0x0a	Line Feed
\v	0x0b	Vertical Tab
\f	0x0c	Form Feed
\r	0x0d	Carriage Return
\e	0x1b	Escape
\G	0x1d	Group Selector
\R	0x1e	Record Selector

Input data can be read directly from file using the -i switch as shown below. The input file is assumed to be Unicode (UTF-8) formatted unless an alternative mode is selected.

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

4.2 Directing Output

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

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

draws a Code 128 barcode in the file here.png. If an encapsulated Post Script file is needed simply append the file name with .eps:

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

Scalable Vector Graphics representations of symbols can be generated with the suffix ".svg". Output can also be directed to stdout using the `--directeps`, `--directpng` and `--directsvg` switches for EPS, PNG and SVG output respectively.

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 in the following table.

Numeric Value	Barcode Name
1	Code 11
2	Standard Code 2 of 5
3	Interleaved 2 of 5
4	Code 2 of 5 IATA
6	Code 2 of 5 Data Logic
7	Code 2 of 5 Industrial
8	Code 3 of 9 (Code 39)
9	Extended Code 3 of 9 (Code 39+)
13	EAN
16	GS1-128 (UCC.EAN-128)
18	Codabar
20	Code 128 (automatic subset switching)
21	Deutshe Post Leitcode
22	Deutshe Post Identcode
23	Code 16K
24	Code 49
25	Code 93

28	Flattermarken
29	GS1 DataBar-14
30	GS1 DataBar Limited
31	GS1 DataBar Extended
32	Telepen Alpha
34	UPC A
37	UPC E
40	PostNet
47	MSI Plessey
49	FIM
50	LOGMARS
51	Pharmacode One-Track
52	PZN
53	Pharmacode Two-Track
55	PDF417
56	PDF417 Truncated
57	Maxicode
58	QR Code
60	Code 128 (Subset B)
63	Australia Post Standard Customer
66	Australia Post Reply Paid
67	Australia Post Routing
68	Australia Post Redirection
69	ISBN (EAN-13 with verification stage)
70	Royal Mail 4 State (RM4SCC)
71	Data Matrix
72	EAN-14
75	NVE-18
76	Japanese Postal Code

77	Korea Post
79	GS1 DataBar-14 Stacked
80	GS1 DataBar-14 Stacked Omnidirectional
81	GS1 DataBar Expanded Stacked
82	PLANET
84	MicroPDF417
85	USPS OneCode
86	Plessey Code
87	Telepen Numeric
89	ITF-14
90	Dutch Post KIX Code
92	Aztec Code
93	DAFT Code
97	Micro QR Code
98	HIBC Code 128
99	HIBC Code 39
102	HIBC Data Matrix
104	HIBC QR Code
106	HIBC PDF417
108	HIBC MicroPDF417
112	HIBC Aztec Code
128	Aztec Runes
129	Code 32
130	Composite Symbol with EAN linear component
131	Composite Symbol with GS1-128 linear component
132	Composite Symbol with GS1 DataBar-14 linear component

133	Composite Symbol with GS1 DataBar Limited component
134	Composite Symbol with GS1 DataBar Extended component
135	Composite Symbol with UPC A linear component
136	Composite Symbol with UPC E linear component
137	Composite Symbol with GS1 DataBar-14 Stacked component
138	Composite Symbol with GS1 DataBar-14 Stacked Omnidirectional component
139	Composite Symbol with GS1 DataBar Expanded Stacked component
140	Channel Code
141	Code One
142	Grid Matrix

This table is also accessible from the command line by issuing zint -t

4.4 Adjusting height

The height of a 1d symbol can be adjusted using the --height switch. For example:

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

specifies a symbol height of 100 times the *x-resolution* of the symbol.

4.5 Adjusting whitespace

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

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

specifies a whitespace width of 10 times the *x-resolution* of the symbol.

4.6 Adding boundary bars and boxes

Zint allows the symbol to be bound with 'boundary 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 whitespace with the --box option. This option is automatically selected for ITF-14 symbols.

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

```
zint --box --border=10 -d 'This'
```

gives a box with a width 10 times the *x-resolution* of the symbol.

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 switch allows the default colours to be inverted so that a white symbol is shown on a black background. For example the command

```
zint -r -d 'This'
```

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

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

```
zint --fg=004700 -d 'This'
```

alters the symbol to a dark green as shown below.

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. This option is only available with PNG output.

`--rotate=0` (default)	`--rotate=180`
`--rotate=270`	`--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. For example for PNG images a scale of 5 will increase the x-dimension to 10 pixels.

4.10 Input modes

GS1 data can be encoded in a number of symbologies. Application identifiers are enclosed in [square brackets] followed by the data to be encoded (see 5.1.12.3). To encode GS1 data use `--gs1`. GS1 mode is assumed (and doesn't need to be set) for EAN-128, DataBar and Composite symbologies but is also available for Code 16k, Data Matrix (ECC 200 only), Aztec Code and QR Code.

QR Code, Micro QR Code and Grid Matrix standards can encode Kanji characters. These can be given in Unicode (UTF-8) format as part of the input data string. Conversion from Unicode to Shift-JIS or GB 2312 as appropriate is handled by Zint.

If the input data is not Unicode encoded or should be interpreted as an 8-bit data stream then the --binary option can be used to achieve this.

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 return 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 file name as shown in the table below:

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:

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

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

Additional options are available which are specific to certain symbologies. These may, for example, control the quantity of error correction data or the size of the symbol. These options are discussed in section 5 of this guide.

5. Using the API

Zint has been written using the C language and currently only has an API for use with C language programs. A wrapper is available for Pascal/Delphi developers thanks to theunknownones.

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 <stdio.h>
#include <zint.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 in the current working directory:

```
#include <stdio.h>
#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 <stdio.h>
```

```
#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;
}
```

Input strings should be Unicode formatted.

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, unsigned char *input, int
length);
int ZBarcode_Encode_File(struct zint_symbol *symbol, char *filename);
int ZBarcode_Print(struct zint_symbol *symbol, int rotate_angle);
int ZBarcode_Encode_and_Print(struct zint_symbol *symbol, unsigned char *input,
int length, int rotate_angle);
int ZBarcode_Encode_File_and_Print(struct zint_symbol *symbol, 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 NULL (ASCII 0) characters in those symbologies which allow this. A value of 0 will disable this function and Zint will encode data up to the first NULL character in the input string.

The "rotate_angle" value can be used to rotate the image when outputting as a PNG image. 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 file where the filename is given in the "filename" string.

5.4 Buffering Symbols in Memory

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 ZBarcide_Encode_and_Buffer(struct zint_symbol *symbol, unsigned char *input,
int length, int rotate_angle);
int ZBarcode_Encode_File_and_Buffer(struct zint_symbol *symbol, char *filename,
int rotate_angle);
```

The arguments here are the same as above. The difference is that instead of saving the image to file it is placed in a character array. The "bitmap" pointer (see below) 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 at the same time as using the buffer functions in the same way as when saving to a PNG image. The bitmap data can be extracted from the character 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 implimented by the external application:

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

for(row = 0; row < my_symbol->bitmap_height; row++) {
        for(column = 0; column < my_symbol->bitmap_width; column++) {
                red = my_symbol->bitmap[i];
```

```
                green = my_symbol->bitmap[i + 1];
                blue = my_symbol->bitmap[i + 2];
                render_pixel(row, column, red, green, blue);
                i += 3;
        }
}
```

5.5 Setting Options

So far our application is not very useful unless we plan to only make Code 128 barcodes and we don't mind that they only save to out.png. As with the front end 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:

Variable Name	Type	Meaning	Default Value
`symbology`	integer	Symbology to use (see section 5.7).	`BARCODE_CODE128`
`height`	integer	Symbol height. [1]	50
`whitespace_width`	integer	Whitespace width.	0
`boder_width`	integer	Border width.	0
`output_options`	integer	Binding or box parameters (see section 5.8). [2]	(none)

`fgcolour`	character string	Foreground (ink) colour as RGB hexadecimal string. Must be 6 characters followed by terminating \ 0 character.	`"000000"`
`bgcolour`	character string	Background (paper) colour as RGB hexadecimal string. Must be 6 characters followed by terminating \ 0 character.	`"ffffff"`
`outfile`	character string	Contains the name of the file to output a resulting barcode symbol to. Must end in .png, .eps or .svg	`"out.png"`
`option_1`	integer	Symbology specific options.	(automatic)
`option_2`	integer	Symbology specific options.	(automatic)
`option_3`	integer	Symbology specific options.	(automatic)
`scale`	float	Scale factor for adjusting size of image.	1.0
`input_mode`	integer	Set encoding of input data (see section 5.9)	`BINARY_MODE`

primary	character string	Primary message data for more complex symbols.	NULL
text	unsigned character string	Human readable text, which usually consists of the input data plus one or more check digits. Uses UTF-8 formatting.	NULL
rows	integer	Number of rows used by the symbol or, if using barcode stacking, the row to be used by the next symbol.	(output only)
width	integer	Width of the generated symbol.	(output only)
encoding_data	array of character strings	Representation of the encoded data.	(output only)
row_height	array of integers	Representation of the height of a row.	(output only)
errtxt	character string	Error message in the event that an error occurred.	(output only)
bitmap	pointer to 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)

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 <stdio.h>
#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;
        ZBarcode_Encode_and_Print(my_symbol, argv[1], 0, 0);
        ZBarcode_Delete(my_symbol);
        return 0;
}
```

5.6 Handling Errors

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

Return Value	Meaning
WARN_INVALID_OPTION	One of the values in `zint_struct` was set incorrectly but Zint has made a guess at what it should have been and generated a barcode accordingly.
ERROR_TOO_LONG	The input data is too long or too short for the selected symbology. No symbol has been generated.
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.
ERROR_INVALID_CHECK	An ISBN with an incorrect check digit has been entered. No symbol has been generated.
ERROR_INVALID_OPTION	One of the values in `zint_struct` was set incorrectly and Zint was unable to guess what it should have been. No symbol has been generated.
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.
ERROR_FILE_ACCESS	Zint was unable to open the requested output file. This is usually a file permissions problem.
ERROR_MEMORY	Zint ran out of memory. This should only be a problem with legacy systems.

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

```
#include <stdio.h>
#include <zint.h>
#include <string.h>
int main(int argc, char **argv)
{
        struct zint_symbol *my_symbol;
        int error = 0;
        my_symbol = ZBarcode_Create();
        strcpy(my_symbol->fgcolour, "nonsense");
        error = ZBarcode_Encode_and_Print(my_symbol, argv[1], 0, 0);
        if(error != 0)
        {
                /* some error occurred */
                printf("%s\n", my_symbol->errtxt);
        }
        if(error > WARN_INVALID_OPTION)
        {
                /* 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: malformed foreground colour target
```

5.7 Specifying a Symbology

Symbologies can be specified by number or by name as shown in the following table. For example

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

means the same as

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

Numeric Value	Name	Symbology
1	BARCODE_CODE11	Code 11
2	BARCODE_C25MATRIX	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
16	BARCODE_EAN128	GS1-128
18	BARCODE_CODABAR	Codabar
20	BARCODE_CODE128	Code 128 (automatic subset switching)
21	BARCODE_DPLEIT	Deutshe Post Leitcode
22	BARCODE_DPIDENT	Deutshe Post Identcode
23	BARCODE_CODE16K	Code 16K
24	BARCODE_CODE49	Code 49
25	BARCODE_CODE93	Code 93
28	BARCODE_FLAT	Flattermarken
29	BARCODE_RSS14	GS1 DataBar-14
30	BARCODE_RSS_LTD	GS1 DataBar Limited
31	BARCODE_RSS_EXP	GS1 DataBar Expanded

32	BARCODE_TELEPEN	Telepen Alpha
34	BARCODE_UPCA	UPC A
37	BARCODE_UPCE	UPC E
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_PDF417TRUNC	PDF417 Truncated
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_AUSREDIRECT	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
72	BARCODE_EAN14	EAN-14
75	BARCODE_NVE18	NVE-18
76	BARCODE_JAPANPOST	Japanese Post
77	BARCODE_KOREAPOST	Korea Post
79	BARCODE_RSS14STACK	GS1 DataBar-14 Stacked
80	BARCODE_RSS14STACK_OMNI	GS1 DataBar-14 Stacked Omnidirectional
81	BARCODE_RSS_EXPSTACK	GS1 DataBar Expanded Stacked

82	BARCODE_PLANET	PLANET
84	BARCODE_MICROPDF417	MicroPDF417
85	BARCODE_ONECODE	USPS OneCode
86	BARCODE_PLESSEY	Plessey Code
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
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
104	BARCODE_HIBC_QR	HIBC QR Code
106	BARCODE_HIBC_PDF	HIBC PDF417
108	BARCODE_HIBC_MICPDF	HIBC MicroPDF417
112	BARCODE_HIBC_AZTEC	HIBC Aztec Code
128	BARCODE_AZRUNE	Aztec Runes
129	BARCODE_CODE32	Code 32
130	BARCODE_EANX_CC	Composite Symbol with EAN linear component
131	BARCODE_EAN128_CC	Composite Symbol with GS1-128 linear component
132	BARCODE_RSS14_CC	Composite Symbol with GS1 DataBar-14 linear component
133	BARCODE_RSS_LTD_CC	Composite Symbol with GS1 DataBar Limited component
134	BARCODE_RSS_EXP_CC	Composite Symbol with GS1 DataBar Extended 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_RSS14STACK_CC	Composite Symbol with GS1 DataBar-14 Stacked component
138	BARCODE_RSS14_OMNI_CC	Composite Symbol with GS1 DataBar-14 Stacked Omnidirectional component
139	BARCODE_RSS_EXPSTACK_CC	Composite Symbol with GS1 DataBar Expanded Stacked component
140	BARCODE_CHANNEL	Channel Code
141	BARCODE_CODEONE	Code One
142	BARCODE_GRIDMATRIX	Grid Matrix

5.8 Adding Boxes and Boundary Bars

Boxes and boundary bars are handled using the `output_options` variable in the `zint_symbol` structure. To use this option simply assign a value to the `output_options` variable from the following table [2].

Value	Effect
0	No box or boundary bars.
BARCODE_BIND	Boundary bars above and below the symbol and between rows if stacking multiple symbols.
BARCODE_BOX	Add a box surrounding the symbol and whitespace.

5.9 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.

Value	Effect
DATA_MODE	Uses full ASCII range interpreted as Latin-1 or binary data.
UNICODE_MODE	Uses pre-formatted UTF-8 input.
GS1_MODE	Encodes GS1 data using FNC1 characters.

5.10 Verifying Symbology Availability

An additional function available in the API is defined as:

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

This function allows you to check whether a given symbology is available. A non-zero return value indicates that the given symbology is available. For example:

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

[1] This value is ignored for Australia Post 4-State Barcodes, PostNet, PLANET, USPS OneCode, RM4SCC, PDF417, Data Matrix, Maxicode, QR Code, GS1 DataBar-14 Stacked, PDF417 and MicroPDF417 - all of which have a fixed height.

[2] This value is ignored for Code 16k and ITF-14 symbols.

6 Types of Symbology

6.1 One-Dimensional Symbols

One-Dimensional 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

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 any length string consisting of the digits 0-9 and the dash character (-). One modulo-11 check digit is calculated.

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

Also known as Code 2 of 5 Matrix is a self-checking code used in industrial applications and photo development. Standard Code 2 of 5 will encode any length numeric input (digits 0-9).

6.1.2.2 IATA Code 2 of 5

Used for baggage handling in the air-transport industry by the International Air Transport Agency, this self-checking code will encode any length numeric input (digits 0-9) and does not include a check digit.

6.1.2.3 Industrial Code 2 of 5

Industrial Code 2 of 5 can encode any length numeric input (digits 0-9) and does not include a check digit.

6.1.2.4 Interleaved Code 2 of 5

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. No check digit is added.

6.1.2.5 Code 2 of 5 Data Logic

Data Logic does not include a check digit and can encode any length numeric input (digits 0-9).

6.1.2.6 ITF-14

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 calculated.

6.1.2.7 Deutsche Post Leitcode

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

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 (EN 797)

6.1.3.1 UPC Version A

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 --barcode=34 -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");
```

6.1.3.2 UPC Version E

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 --barcode=37 -d 1123456
```

or

```
my_symbol->symbology = BARCODE_UPCE;
error = ZBarcode_Encode_and_Print(my_symbol, "1123456");
```

6.1.4 European Article Number (EN 797)

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

The EAN system is used in retail across Europe and includes standards for EAN-2 and EAN-5 add-on codes, 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 using the + symbol as with UPC symbols. For example

```
zint --barcode=13 -d 54321
```

will encode a stand-alone EAN-5, whereas

```
zint --barcode=13 -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");
  error = ZBarcode_Encode_and_Print(my_symbol, "7432365+54321");
```

All of the EAN symbols include check data which is added by Zint.

6.1.4.2 SBN, ISBN and ISBN-13

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 + symbol as with UPC symbols.

6.1.5 Plessey

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

6.1.6 MSI Plessey

Based on Plessey and developed by MSE Data Corporation, MSI Plessey is available with a range of check digit options available by setting `option_2` or by using the `--ver=` switch. Any length numeric (digits 0-9) input can be encoded. The table below shows the options available:

Value of `option_2`	Check Digits
0	None
1	Modulo-10
2	Modulo-10 & Modulo-10
3	Modulo-11
4	Modulo-11 & Modulo-10

6.1.7 Telepen

6.1.7.1 Telepen Alpha

Telepen Alpha was developed by SB Electronic Systems Limited and can encode any length of ASCII text input. Telepen includes a modulo-127 check digit.

6.1.7.2 Telepen Numeric

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). Includes a modulo-127 check digit.

6.1.8 Code 39

6.1.8.1 Standard Code 39 (ISO 16388)

Standard Code 39 was developed in 1974 by Intermec. Input data can be of any length and supports 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 `option_2 = 1` or using `--ver=1.`

6.1.8.2 Extended Code 39

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

6.1.8.3 Code 93

A variation of Extended Code 39, Code 93 also supports full ASCII text. Two check digits are added by Zint.

6.1.8.4 PZN

PZN is a Code 39 based symbology used by the pharmaceutical industry in Germany. PZN encodes a 6 digit number and includes a modulo-10 check digit.

6.1.8.5 LOGMARS

LOGMARS (Logistics Applications of Automated Marking and Reading Symbols) is a variation of the Code 39 symbology used by the US Department of Defence. LOGMARS encodes the same character set as Standard Code 39 and adds a modulo-43 check digit.

6.1.8.6 Code 32

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

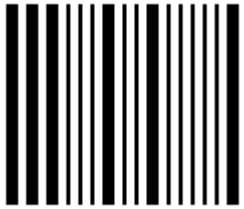
6.1.8.7 HIBC Code 39

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.

6.1.9 Codabar (EN 798)

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 any length string 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 digit is generated.

6.1.10 Pharmacode



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

6.1.11 Code 128

6.1.11.1 Standard Code 128 (ISO 15417)

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 Latin-1 (non-English) characters in Code 128 symbols [1]. The Latin-1 character set is shown in Appendix A.

6.1.11.2 Code 128 Subset B

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

6.1.11.3 GS1-128

A variation of Code 128 also known as UCC/EAN-128, this symbology is defined by the GS1 General Specification. Application Identifiers (AIs) should be entered using [square brackets] notation. These will be converted to (round brackets) for the human readable text. This will allow round brackets to be used in the data strings to be encoded. Fixed length data should be entered at the appropriate length for correct encoding (see Appendix C). GS1-128 does not support extended ASCII characters. Check digits for GTIN data (AI 01) are not generated and need to be included in input data. The following is an example of a valid GS1-128 input:

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

6.1.11.4 EAN-14

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

6.1.11.5 NVE-18

A variation of Code 128 the Nummber der Versandeinheit standard includes both modulo-10 and modulo-103 check digits. NVE-18 requires a 17 digit numerical input.

6.1.11.6 HIBC Code 128

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.

6.1.12 GS1 DataBar (ISO 24724)

Also known as RSS (Reduced Spaced Symbology) these symbols are due to replace GS1-128 symbols starting in 2010 in accordance with the GS1 General Specification. If a GS1 DataBar symbol is to be printed with a 2D component as specified in ISO 24723 set `option_1 = 2` or use the option `--mode=2` at the command prompt. See section 6.3 of this manual to find out how to generate DataBar symbols with 2D components.

6.1.12.1 DataBar-14 and DataBar-14 Truncated

Also known as RSS-14 this standard encodes a 13 digit item code. A check digit and application identifier of (01) are added by Zint. Note that for full standard compliance symbol height should be greater than or equal to 33 modules. For DataBar-14 Truncated set the symbol height to a minimum of 13.

6.1.12.2 DataBar Limited

Also known as RSS Limited this standard encodes a 13 digit item code and can be used in the same way as DataBar-14 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-14 a check digit and application identifier of (01) are added by Zint.

6.1.12.3 DataBar Expanded

Also 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 (rounded brackets) before it is included in the human readable text attached to the symbol. This method allows the inclusion of rounded brackets in the data to be encoded. 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 (see Appendix C). The following is an example of a valid DataBar Expanded input

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

6.1.13 Korea Post Barcode

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

6.1.14 Channel Code

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 `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:

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

Note that 7 and 8 channel codes require a processor intensive algorithm to generate and so response times when generating these codes will be relatively slow.

6.2 Stacked Symbologies

6.2.1 Basic Symbol Stacking

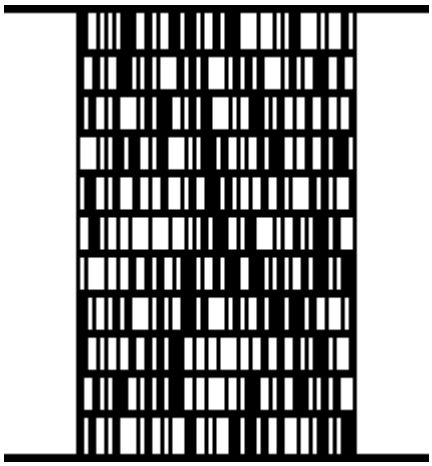
An early innovation to get more information into a symbol, used primarily in the vehicle industry, is to simply stack one-dimensional codes on top of each other. This can be achieved at the command prompt by giving more than one set of input data. For example

```
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");
error = ZBarcode_Encode(my_symbol, "That");
error = ZBarcode_Print(my_symbol);
```

The example below shows 5 EAN-13 symbols stacked in this way.



6.2.2 Code 16k (EN 12323)

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 Code 16k which uses a Code128 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.

6.2.3 PDF417 (ISO 15438)

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 followed by a number between 1 and 30, and the amount of check digit information can be specified by using the --security switch followed by a number between 0 and 8 where the number of codewords used for check information is determined by $2^{(value + 1)}$. If using the API these values are assigned to `option_2` and `option_1` respectively. The default level of check information is determined by the amount of data being encoded. International text support is provided using the Latin-1 character set as described in Appendix A. A separate symbology ID 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.

6.2.4 Compact PDF417

Also known as truncated PDF417. Options are as for PDF417 above.

6.2.5 MicroPDF417 (ISO 24728)

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

6.2.6 GS1 DataBar-14 Stacked (ISO 24724)

A stacked variation of the GS1 DataBar-14 symbol requiring the same input (see section 6.1.12.1). The height of this symbol is fixed. 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.

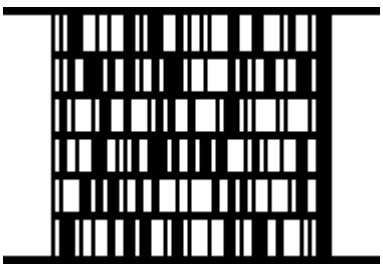
6.2.7 GS1 DataBar-14 Stacked Omnidirectional (ISO 24724)

Another variation of the GS1 DataBar-14 symbol requiring the same input (see section 6.1.12.1). 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.

6.2.8 GS1 DataBar Expanded Stacked (ISO 24724)

A stacked variation of the GS1 DataBar Expanded symbol for smaller packages. Input is the same as for GS1 DataBar Expanded (see section 6.1.12.3). In addition the width of the symbol can be altered using the `--cols` switch or `option_2`. In this case the number of columns relates to the number of character pairs on each row of the symbol. For symbols with a 2D component the number of columns must be at least 2. This symbol can be generated with a two-dimensional component to make a composite symbol.

6.2.9 Code 49



Developed in 1987 at Intermec, Code 49 is a cross between UPC and Code 39. It it 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.

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 Specification. 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 or by adjusting `symbol->symbology` as with other encoding methods. Valid values are shown below.

Numeric Value	Name	Symbology
130	BARCODE_EANX_CC	Composite Symbol with EAN linear component
131	BARCODE_EAN128_CC	Composite Symbol with GS1-128 linear component
132	BARCODE_RSS14_CC	Composite Symbol with GS1 DataBar-14 linear component
133	BARCODE_RSS_LTD_CC	Composite Symbol with GS1 DataBar Limited component
134	BARCODE_RSS_EXP_CC	Composite Symbol with GS1 DataBar Extended 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_RSS14STACK_CC	Composite Symbol with GS1 DataBar-14 Stacked component
138	BARCODE_RSS14_OMNI_CC	Composite Symbol with GS1 DataBar-14 Stacked Omnidirectional component
139	BARCODE_RSS_EXPSTACK_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=` command. For example:

```
zint -b 130 --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 = 130;my_symbol->option_1 = 1;
```

```
strcpy(my_symbol->primary, "331234567890");
ZBarcode_Encode_and_Print(my_symbol, "[99]1234-abcd");
```

EAN-2 and EAN-5 add-on data can be used with EAN and UPC symbols using the + symbol as described in section 6.1.3 and 5.1.4.

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 dependant on the length of the input string. Alternatively the three methods can be accessed using the `--mode=` prompt followed by 1, 2 or 3 for CC-A, CC-B or CC-C respectively, or by using the `option_1` variable as shown above.

6.3.1 CC-A

This system uses a variation of MicroPDF417 which 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`.

6.3.2 CC-B

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`.

6.3.3 CC-C

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`.

6.4 Two-Track Symbols

6.4.1 Two-Track Pharmacode

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

6.4.2 PostNet

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 any length, standard lengths as used by USPS were PostNet6 (5 digits ZIP input), PostNet10 (5 digit ZIP + 4 digit user data) and PostNet12 (5 digit ZIP + 6 digit user data).

6.4.3 PLANET

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 any 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

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:

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	99999999AAAAAAAAAA	67-bar	62	C
23	99999999999999999999999	67-bar	62	N

6.5.1.2 Reply Paid Barcode

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

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

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

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 Country Code (RM4SCC)

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 Picadilly Circus in London. Check digit data is generated by Zint.

6.5.4 USPS OneCode

Also known as the Intelligent Mail Barcode and used in the US by the United States Postal Service (USPS), the OneCode system replaced the PostNet and PLANET symbologies in 2009. OneCode 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"
```

6.5.5 Japanese Postal Code

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.

6.6 Two-Dimensional Symbols

6.6.1 Data Matrix (ISO 16022)

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 can encode any characters in the Latin-1 set and can also encode GS1 data. The size of the generated symbol can also be adjusted using the `--vers=` option or by setting `option_2` as shown in the table below. A separate symbology ID 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. Note that only ECC200 encoding is supported, the older standards have now been removed from Zint.

Input	Symbol Size	Input	Symbol Size
1	10 x 10	16	64 x 64
2	12 x 12	17	72 x 72
3	14 x 14	18	80 x 80
4	16 x 16	19	88 x 88
5	18 x 18	20	96 x 96
6	20 x 20	21	104 x 104
7	22 x 22	22	120 x 120
8	24 x 24	23	132 x 132
9	26 x 26	24	144 x 144
10	32 x 32	25	8 x 18
11	36 x 36	26	8 x 32
12	40 x 40	27	12 x 26
13	44 x 44	28	12 x 36
14	48 x 48	29	16 x 36
15	52 x 52	30	16 x 48

An extra feature is available for Data Matrix symbols which allows Zint to automatically resize the symbol as required but also prevents Zint from using rectangular symbols. To set this mode at the command line use the option `--square` and when using the API set the value `option_3 = DM_SQUARE.`

6.6.2 QR Code (ISO 18004)

Also known as Quick Response Code this symbology was developed by Denso.
Four levels of error correction are available using the `security=` option or setting `option_1` as shown in the following table.

Input	ECC Level	Error Correction Capacity	Recovery Capacity
1	L (default)	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 set by using the `vers=` option or setting `option_2` to the QR Code version required (1-40). The size of symbol generated is shown in the table below.

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

19	93 x 93	39	173 x 173
20	97 x 97	40	177 x 177

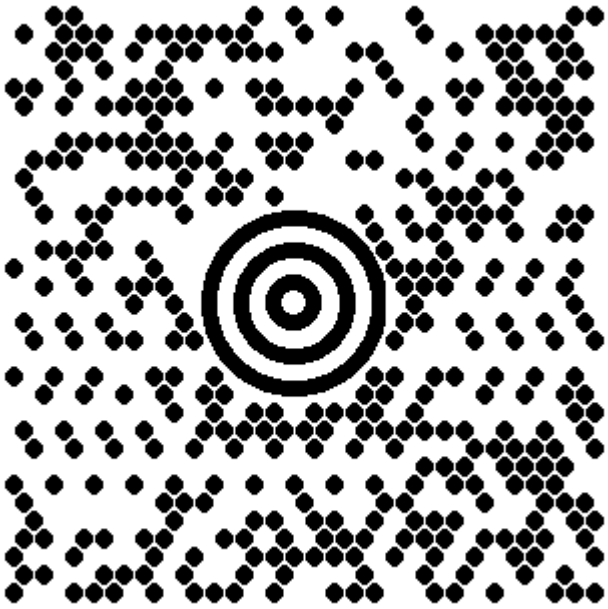
The maximum capacity of a (version 40) QR Code symbol is 7089 numeric digits, 4296 alphanumeric characters or 2953 bytes of data. QR Code symbols can also be used to encode GS1 data. QR Code symbols can encode characters in the Latin-1 set and Kanji characters which are members of the Shift-JIS encoding scheme. Input should be entered as a UTF-8 stream with conversion to Shift-JIS being carried out automatically by Zint. A separate symbology ID 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.

6.6.3 Micro QR Code (ISO 18004)

A miniature version of the QR Code symbol for short messages. ECC levels can be selected as for QR Code (above). QR Code symbols can encode characters in the Latin-1 set and Kanji characters which are members of the Shift-JIS encoding scheme. Input should be entered as a UTF-8 stream with conversion to Shift-JIS beingcarried out automatically by Zint. A preferred symbol size can be selected by using the `vers=` option or by setting `option_2` although the actual version used by Zint may be different if required by the input data. The table below shows the possible sizes:

Input	Version	Symbol Size
1	M1	11 x 11
2	M2	13 x 13
3	M3	15 x 15
4	M4	17 x 17

6.6.4 Maxicode (ISO 16023)



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 data field 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:

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 should be filled with the SPACE character (ASCII 32).
10-12	Three digit country code according to ISO 3166 (see Appendix B).
13-15	Three digit service code. This depends on your parcel courier.

The primary message can be designated at the command prompt using the `--primary=` switch. The secondary message uses the normal data entry method. For example:

```
zint -o test.eps -b 57 --primary='999999999840012' -d 'Secondary Message Here'
```

When using the API the primary message must be placed in the `symbol->primary` string. The secondary is entered in the same way as described in section 5.2. When either of these modes is selected Zint will analyse the primary message and select either mode 2 or mode 3 as appropriate.

Modes 4 to 6 can be accessed using the `--mode=` switch or by setting `option_1`. Modes 4 to 6 do not require 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.

All modes support extended ASCII characters and number compression. 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:

Mode	Maximum Data Length for Capital Letters	Maximum Data Length for Numeric Digits	Number of Error Correction Codewords
2 (secondary only)	84	126	50
3 (secondary only)	84	126	50
4	93	135	50
5	77	110	66
6	93	135	50

6.6.5 Aztec Code (ISO 24778)

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 `--ver=` option or setting `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.

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

by setting `option_1` to a value from the following table:

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 is able to encode any extended ASCII character data up to a maximum length of approximately 3823 numeric or 3067 alphabetic characters or 1914 bytes of data. A separate symbology ID 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.

6.6.6 Aztec Runes

A truncated version of compact Aztec Code for encoding whole integers between 0 and 255. Includes Reed-Solomon error correction. As defined in ISO/IEC 24778 Annex A.

6.6.7 Code One

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. There are two types of Code One symbol - variable height symbols which are roughly square (versions A thought to H) and fixed-height versions (version S and T). These can be selected by using `--vers=` or setting `option_2` as shown in the table below:

Input	Version	Size	Numeric Data Capacity	Alphanumeric Data Capacity
1	A	16 x 18	22	13
2	B	22 x 22	44	27
3	C	28 x 32	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	8X height	18	n/a

10	T	16X height	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.

6.6.8 Grid Matrix

The most recently developed encoding standard to be supported by Zint, Grid Matrix became an AIM standard in December 2008. The encoding allows Latin-1 and Chinese characters within the GB 2312 standard set to be encoded in a checkerboard pattern. Input should be entered as a UTF-8 stream with conversion to GB 2312 being carried out automatically by Zint. 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 ver= option or by setting option_2, and the error correction capacity can be specified by using the security= option or by setting option_1 according to the following tables:

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

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

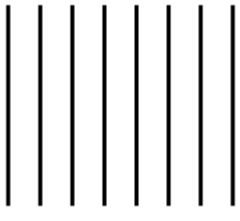
6.7 Other Barcode-Like Markings

6.7.1. Facing Identification Mark (FIM)

Used by the United States Postal Service (USPS), the FIM symbology is used to assist automated mail processing. There are only 4 valid symbols which can be generated using the characters A-D as shown in the table below.

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.

6.7.2 Flattermarken



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 any length numeric data and does not include a check digit.

6.7.3 DAFT Code

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 ignored.

7. Legal and Version Information

7.1 License

Zint, *libzint* and Zint Barcode Studio are Copyright © 2011 Robin Stuart and are distributed under the terms of the GNU General Public License version 3 or later. See the file `COPYING` for more information. The following terms form part of the GPL:

```
This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License as
published by the Free Software Foundation; either version 3 of the
License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
General Public License for more details.
```

Qt4 code is Copyright © 2009 BogDan Vatra, used under the terms of the GNU General Public Licence.

Some Data Matrix code is Copyright © 2004 Adrian Kennard, Andrews & Arnold Ltd and © 2006 Stefan Schmidt, used under the terms of the GNU General Public Licence.

Reed-Solomon code is Copyright © 2004 Cliff Hones, used under the terms of the GNU General Public Licence.

Portions of GS1 DataBar and Composite Symbology code are Copyright © 2006 ISO/BSI Global and used with permission.

Portions of OneCode code is © 2006 United States Postal Service. This is indicated where appropriate in the source code and is used on the understanding that this code has been released to the public domain and that such use is intended by the copyright holder.

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.

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. If, however, you are a patent holder or hold any other intellectual property rights on the methods used by Zint or the symbologies which Zint generates, and do not want Zint to continue to support your symbology then please contact me and I will update the code to accommodate your

wishes at the soonest opportunity.

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

v0.1 - (as *Zebar*) Draws UPC-A. UPC-E, EAN-8, EAN-13, Interlaced 2 of 5, Codabar, Code 39, Extended Code 39 and Code 93 barcodes and Add-on codes EAN-2 and EAN-5 without parity. 13/11/2006

v0.2 - Added Code 128 (which is now the default), Code 11, Code 2 of 5, Add-on codes EAN-2 and EAN-5 parity and MSI/Plessey without check digit. 12/12/2006

v0.3 - Added MSI/Plessey Mod 10 check and 2 x Mod 10 check options, Telepen ASCII and Telepen numeric, Postnet, RM4SCC. Code has been tidied up quite a bit. Bind option added. 30/12/2006

v0.4 - Added barcode stacking (now stacks up to 16 barcodes) and Code16k (stub). 15/1/2007

v0.5 - Added Australia Post 4-State Barcodes and Pharmacode (1 and 2 track). 4-state codes now draw with correct height/width ratio. 28/2/2007

v0.6 - Added Plessey and some derivative codes (EAN-128, Code 128 subset B, Auspost Reply, Auspost Routing, Auspost Redirect, ITF-14). Tidied up code again: separated symbologies into more files and put all lookup tables into arrays (much reducing the amount of code, especially for Code 39e and Code 93). Errors now output to stderr. Added proper input verification. Distribution now packs with HTML pages instead of separate README. Outputs to PNG. Outputs colour. User variable whitespace and border width. Box option. Fixed EAN add-on bug. Added whitespace and height options. Project name changed to Zint to avoid conflict with extant trade name. Added escape character input. 1/4/2007

v1.0 - Corrected problem with escape character codes. Supports PDF417. This completes the list of features I originally wanted (plus a few more), hence skip to version 1.0. 20/4/2007

v1.1 - Added more derivatives (Code 2 of 5 Matrix, IATA and Data Logic, Truncated PDF417, Deutsche Post Leitcode and Identcode, Pharmazentralnummer, Planet) and Flattermarken. Tidied up 2 of 5 code. 26/4/2007

v1.2 - Supports Data Matrix (by absorption of IEC16022 code by Stefan Schmidt et al). Added reverse colours, FIM, MSI/Plessey Modulo 11 and Modulo 11/10. Corrected Code 16k check digit calculation. 28/5/2007

v1.3 - Supports USPS OneCode and LOGMARS. Brought all usage information into one User Manual document. 13/6/2007

v1.4 - Added NVE-18 support. Corrected some problems with compilation and input verification. Command line option handling now uses getopt(), and all the switches have changed. Added --font option. 20/6/2007

v1.5 - Pulled everything together to make an API. Corrected errors with EAN-13, PDF417 and LOGMARS. Added EPS output. Added QR Code support using libqrencode. Corrected ISBN verification error. Re-compiled documentation in HTML form. Put in place proper error handling

routines. --font option removed. Encoding is now done with a restructured zint_symbol structure. Added make install option and optional QR Code support to Makefile. Corrected minor problem with 4-State Codes. Restructured code into fewer source code files. Added MicroPDF417 support. 12/8/2007

v1.5.1 - Added formatting code to EPS output of EAN and UPC symbols according to EN 797:1996. Checked against and, where appropriate, altered or corrected to comply with ISO 16388 and ISO 15417 including Latin-1 support. Altered default image settings, added automatic ITF border. Corrected error with USPS OneCode. Tidied up Code 39 quite a bit, added Mod 43 options. 3/9/2007

v1.5.2 - Added extended ASCII support to Code 16k. Corrected Code 128 error. Added Maxicode support by integrating code by John Lien. 26/9/2007

v1.5.3 - Made huge corrections to Maxicode support by removing and re-writing much of John's code. Maxicode now supports extended ASCII and modes 4, 5 and 6. 10/10/2007

v1.5.4 - Added GS1 DataBar (Reduced Space Symbology) support. 26/11/2007

v1.5.5 - Added composite symbology support. Corrected errors with GS1-128 and PDF417/MicroPDF417 byte processing. Transferred licence to GPL version 3. 9/3/2008

v1.6 - Data Matrix, Maxicode and Australia Post now use common Reed-Solomon functions - this also fixes a bug in Maxicode error correction and replaces the last of the Lien code. Added PNG output for Maxicode symbols. Removed some useless code. Updated QR support for *libqrencode* v2.0.0. 22/4/2008

v1.6.1 - Major restructuring of PNG generating code: Now draws UPCA and EAN symbols properly and puts human readable text into the image. Also corrected some nasty 'never ending loop' bugs in Code 128 and check digit bugs in PostNet and Planet. 8/7/2008

v1.6.2 - Added KIX Code support and PNG image rotation. Corrected a bug affecting extended ASCII support in Code 128 and Code 16k. 28/7/2008.

v2.0 beta - Added support for Aztec Code, Codablock-F, Code 32, EAN-14 and DAFT Code. Re-arranged symbology numbers to match Tbarcode v8. Corrected a never ending loop bug in EAN-128. 29/9/2008

v2.0 beta r2 - Many corrections and bugfixes. (Code 11, Code 128, EAN-128, Aztec Code, Codablock-F, Code 16k, Postnet, PLANET, NVE-18, PZN, Data Matrix, Maxicode and QR Code)

v2.0 - Made corrections to Aztec Code and tested output with bcTester. Added Aztec Runes, Micro QR Code and Data Matrix ECC 000-140. Updated e-mail information. 18/11/2008

v2.1 - Reinstated Korea Post barcodes, harmonised bind and box options, moved Unicode handling into backend and added input_mode option, added size options to Data Matrix, added NULL character handling for Codablock-F, Code 128, Code 16k, Extended Code 39, Code 93, Telepen, Maxicode, Data Matrix ECC 200, PDF417 and MicroPDF417. Added GS1 support for Code 16k, Codablock-F and Aztec Code. Added scale and direct to `stdout` options. Rebult Data Matrix ECC 200 encoding algorithms to support NULL encoding and GS1 data encoding. 31/1/2009

v2.1.1 - Minor Data Matrix bugfix and added HIBC options. 10/2/2009

v2.1.2 - Added SVG output option. Improved Japanese character support including Unicode >

Shift-JIS capability. Bugfixes for Data Matrix (missing characters at end of string) and Codablock-F (K1/K2 check digit and row indicators above row 6). 1/3/2009

v2.1.3 - Many improvements to the QZint GUI which is now renamed "Zint Barcode Studio 0.2". Added Japanese Postal Barcode, Code 49 and Channel Code and made corrections to Data Matrix (Binary mode data compression terminates too soon), Aztec Code (Bug when automatically resizing after removing "all 0" and "all 1" codewords) and Code 128 (Extended ASCII characters become corrupt). 19/5/2009

v2.1.4 - Many stability improvements including removal of buffer overruns in Code 39, LOGMARS, PZN, Aztec Code and Composite CC-A. Addition of files for compiling on MS Windows platform - tested successfully on XP and Vista. 19/6/2009

v2.2 - Added Code One and GS1 support in Code 49. Changed GUI binary name to zint-qt and brought GUI up to version 1.0. Made some minor bugfixes to Code 39, ITF-14, Aztec Code, Code 128 and Code 16K. Added 'rest' button to GUI. Included .spec file from Radist. 18/7/2009

v2.2.1 - Data encoding bugfixes for Aztec Code, Data Matrix, USPS One Code and PDF417. Symbol plotting bugfixes for MicroPDF417 and 2D components of composite symbols. Text formatting bugfix for Qt renderer and a couple of compiler fixes for MSVC PNG image output. 6/8/2009

v2.2.2 - A beta release previewing the new API structure. Better NULL character support with "nullchar" value removed. Added loading from file and sequence dialogs in Barcode Studio. 29/9/2009

v2.3 - Fixed problems with Micro QR Code and rebuilt QR Code support removing dependence on libqrencode. Improved Kanji character support for QR Code and Micro QR Code which now auto-detects and automatically converts to Shift-JIS. Added Grid Matrix symbology with Kanji character support and automatic conversion to GB 2312. Removed no_qr compile option. Advanced Barcode Studio version number to match library version number. 23/11/2009

v2.3.1 - Removed Codablock-F. Redesigned scale function so that human readable text and Maxicode symbols can be scaled consistently. Corrected encoding bugs with Code 128/Code 16k and Data Matrix ECC 050. Added --notext option to CLI. 7/3/2010

v2.3.2 - Corrected many bugs in GS1 DataBar Extended thanks to the careful study of the code by Pablo Orduña at the PIRAmIDE project. Similarly corrected some bugs in Maxicode thanks to Monica Swanson at Source Technologies. Also minor bugfixes for ISBN and Aztec Code, and added some small features like a --square option in the CLI. 29/5/2010

v2.4 - Built extensions to the API for integrating with glabels with thanks to Sam Lown and Jim Evins. Added code optimisation and input from stdin thanks to Ismael Luceno. Reinstated escape character input. Simplification of Barcode Studio. 13/9/2010

v2.4.1 & 2.4.2 – A whole host of bugfixes including correction of ECC routines for Code-1 and addition of batch processing at the command line.

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

Austalia 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 decoding library

7.5 Standard Compliance

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

BS EN 797:1996 Bar coding - Symbology specifications - 'EAN/UPC'

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

BS ISO/IEC 12323:2005 AIDC technologies - Symbology specifications - Code 16K

BS ISO/IEC 15417:2007 Information technology - Automatic identification and data capture techniques - Code 128 bar code symbology specification

BS ISO/IEC 15438:2006 Information technology - Automatic identification and data capture techniques - PDF417 bar code symbology specification

BS ISO/IEC 16022:2006 Information technology - Automatic identification and data capture techniques - Data Matrix bar code symbology specification

BS ISO/IEC 16023:2000 Information technology - International symbology specification - Maxicode

BS ISO/IEC 16388:2007 Information technology - Automatic identification and data capture techniques - Code 39 bar code symbology specification

BS ISO/IEC 18004:2006 Information technology - Automatic identification and data capture

techniques - QR Code 2005 bar code symbology specification

BS ISO/IEC 24723:2006 Information technology - Automatic identification and data capture techniques - EAN.UCC Composite bar code symbology specification

BS ISO/IEC 24724:2006 Information technology - Automatic identification and data capture techniques - Reduced Space Symbology (RSS) bar code symbology specification

BS 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

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/HIBC 2.3-2009 - The Health Industry Bar Code (HIBC) Supplier Labeling Standard

AIMD014 (v 1.63) - Information technology, Automatic identification and data capture techniques - Bar code symbology specification - Grid Matrix (Released 9th Dec 2008)

GS1 General Specifications Version 8.0

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 character support as shown in the subsequent section.

A.1 ASCII Standard

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

Hex	0	1	2	3	4	5	6	7
0	NULL	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	Y	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 8859-1)

A common extension to the ASCII standard, Latin-1 is used to expand the range of Code 128, PDF417 and other symbols. Input strings should be in Unicode format.

Hex	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0			NBSP	°	À	Đ	à	ð
1			¡	±	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. Three Digit Country Codes (ISO 3166)

Below are some of the three digit country codes as determined by ISO 3166 for use with Maxicode symbols.

AFGHANISTAN	004
ALAND ISLANDS	248
ALBANIA	008
ALGERIA	012
AMERICAN SAMOA	016
ANDORRA	020
ANGOLA	024
ANGUILLA	660
ANTARCTICA	010
ANTIGUA AND BARBUDA	028
ARGENTINA	032
ARMENIA	051
ARUBA	053
AUSTRALIA	036
AUSTRIA	040
AZERBAIJAN	031
BAHAMAS	044
BAHRAIN	048
BANGLADESH	050
BARBADOS	052
BELARUS	112
BELGIUM	056
BELIZE	084
BENIN	204
BERMUDA	060
BHUTAN	064
BOLIVIA	068
BOSNIA AND HERZEGOVINA	070
BOTSWANA	072
BOUVET ISLAND	074
BRAZIL	076
BRITISH INDIAN OCEAN TERRITORY	086
BRUNEI DARUSSALAM	096
BULGARIA	100
BURKINA FASO	854
BURUNDI	108
CAMBODIA	116
CAMEROON	120
CANADA	124
CAPE VERDE	132
CAYMAN ISLANDS	136
CENTRAL AFRICAN REPUBLIC	140
CHAD	148
CHILE	152
CHINA	156
CHRISTMAS ISLAND	162
COCOS (KEELING) ISLANDS	166
COLOMBIA	170
COMOROS	174
CONGO	178

CONGO, THE DEMOCRATIC REPUBLIC OF THE	180
COOK ISLANDS	184
COSTA RICA	188
COTE D'IVOIRE	384
CROATIA	191
CUBA	192
CYPRUS	196
CZECH REPUBLIC	203
DENMARK	208
DJIBOUTI	262
DOMINICA	212
DOMINICAN REPUBLIC	214
ECUADOR	218
EGYPT	818
EL SALVADOR	222
EQUATORIAL GUINEA	226
ERITREA	232
ESTONIA	233
ETHIOPIA	231
FALKLAND ISLANDS (MALVINAS)	238
FAROE ISLANDS	234
FIJI	242
FINLAND	246
FRANCE	250
FRENCH GUIANA	254
FRENCH POLYNESIA	258
FRENCH SOUTHERN TERRITORIES	260
GABON	266
GAMBIA	270
GEORGIA	268
GERMANY	276
GHANA	288
GIBRALTAR	292
GREECE	300
GREENLAND	304
GRENADA	308
GUADELOUPE	312
GUAM	316
GUATEMALA	320
GUERNSEY	831
GUINEA	324
GUINEA-BISSAU	624
GUYANA	328
HAITI	332
HEARD ISLAND AND MCDONALD ISLANDS	334
HOLY SEE (VATICAN CITY STATE)	336
HONDURAS	340
HONG KONG	344
HUNGARY	348
ICELAND	352
INDIA	356
INDONESIA	360
IRAN (ISLAMIC REPUBLIC OF)	364
IRAQ	368
IRELAND	372
ISLE OF MAN	833
ISRAEL	376
ITALY	380
JAMAICA	388
JAPAN	392
JERSEY	832

JORDAN	400
KAZAKHSTAN	398
KENYA	404
KIRIBATI	296
KOREA, DEMOCRATIC PEOPLE'S REPUBLIC OF	408
KOREA, REPUBLIC OF	410
KUWAIT	414
KYRGYZSTAN	417
LAO PEOPLE'S DEMOCRATIC REPUBLIC	418
LATVIA	428
LEBANON	422
LESOTHO	426
LIBERIA	430
LIBYAN ARAB JAMAHIRIYA	434
LIECHTENSTEIN	438
LITHUANIA	440
LUXEMBOURG	442
MACAO	446
MACEDONIA, THE FORMER YUGOSLAV REPUBLIC OF	807
MADAGASCAR	450
MALAWI	454
MALAYSIA	458
MALDIVES	462
MALI	466
MALTA	470
MARSHALL ISLANDS	584
MARTINIQUE	474
MAURITANIA	478
MAURITIUS	480
MAYOTTE	175
MEXICO	484
MICRONESIA, FEDERATED STATES OF	583
MOLDOVA, REPUBLIC OF	498
MONACO	492
MONGOLIA	496
MONTENEGRO	499
MONTSERRAT	500
MOROCCO	504
MOZAMBIQUE	508
MYANMAR	104
NAMIBIA	516
NAURU	520
NEPAL	524
NETHERLANDS	528
NETHERLANDS ANTILLES	530
NEW CALEDONIA	540
NEW ZEALAND	554
NICARAGUA	558
NIGER	562
NIGERIA	566
NIUE	570
NORFOLK ISLAND	574
NORTHERN MARIANA ISLANDS	580
NORWAY	578
OMAN	512
PAKISTAN	586
PALAU	585
PALESTINIAN TERRITORY, OCCUPIED	275
PANAMA	591
PAPUA NEW GUINEA	598
PARAGUAY	600

PERU	604
PHILIPPINES	608
PITCAIRN	612
POLAND	616
PORTUGAL	620
PUERTO RICO	630
QATAR	634
REUNION	638
ROMANIA	642
RUSSIAN FEDERATION	643
RWANDA	646
SAINT HELENA	654
SAINT KITTS AND NEVIS	659
SAINT LUCIA	662
SAINT PIERRE AND MIQUELON	666
SAINT VINCENT AND THE GRENADINES	670
SAMOA	882
SAN MARINO	674
SAO TOME AND PRINCIPE	678
SAUDI ARABIA	682
SENEGAL	686
SERBIA	688
SEYCHELLES	690
SIERRA LEONE	694
SINGAPORE	702
SLOVAKIA	703
SLOVENIA	705
SOLOMON ISLANDS	090
SOMALIA	706
SOUTH AFRICA	710
SOUTH GEORGIA AND THE SOUTH SANDWICH ISLANDS	239
SPAIN	724
SRI LANKA	144
SUDAN	736
SURINAME	740
SVALBARD AND JAN MAYEN	744
ST. HELENA	654
ST. PIERRE AND MIQUELON	666
SWAZILAND	748
SWEDEN	752
SWITZERLAND	756
SYRIAN ARAB REPUBLIC	760
TAIWAN, PROVINCE OF CHINA	158
TAJIKISTAN	762
TANZANIA, UNITED REPUBLIC OF	834
THAILAND	764
TIMOR-LESTE	626
TOGO	768
TOKELAU	772
TONGA	776
TRINIDAD AND TOBAGO	780
TUNISIA	788
TURKEY	792
TURKMENISTAN	795
TURKS AND CAICOS ISLANDS	796
TUVALU	798
UGANDA	800
UKRAINE	804
UNITED ARAB EMIRATES	784
UNITED KINGDOM	826
UNITED STATES	840

```
UNITED STATES MINOR OUTLYING ISLANDS            581
URUGUAY                                         858
UZBEKISTAN                                      860
VANUATU                                         548
VATICAN CITY STATE (HOLY SEE)                   336
VENEZUELA                                       862
VIET NAM                                        704
VIRGIN ISLANDS (BRITISH)                        092
VIRGIN ISLANDS (U.S.)                           850
WALLIS AND FUTUNA                               876
WESTERN SAHARA                                  732
YEMEN                                           887
YUGOSLAVIA                                      891
ZAMBIA                                          894
ZIMBABWE                                        716
```

C. GS1 General Specification

The GS1 General Specification defines a global standard for encoding data about products. The full specification is available here. Data is encoded as a series of number pairs where the first number, usually shown in (brackets) is an application identifier (AI), and the second is a formatted representation of the data. For example (401)6773 can be read as "Consignment Number 6773" where the AI (401) signifies that the data is a consignment number. Note that when using Zint AI data is entered using [square] brackets. This allows rounded brackets to be included in the data which is allowed by the specification. When the barcode symbol is generated these square brackets are replaced by rounded brackets in any text displayed. A list of valid AI numbers is given below.

C.1 Application Identifiers [1]

```
00 Serial Shipping Container Code (SSCC)
01 Global Trade Item Number (GTIN)
02 # of containers10 Batch Number
11 Production Date13 Packaging Date
15 Sell by Date (Quality Control)
17 Expiration Date20 Product Variant
21 Serial Number22 HIBCC Quantity, Date, Batch and Link
23x Lot Number
240 Additional Product Identification
250 Second Serial Number30 Quantity Each
310y Product Net Weight in kg
311y Product Length/1st Dimension, in meters
312y Product Width/Diameter/2nd Dimension, in meters
313y Product Depth/Thickness/3rd Dimension, in meters
314y Product Area, in square meters
315y Product Volume, in liters
316y product Volume, in cubic meters
320y Product Net Weight, in pounds
321y Product Length/1st Dimension, in inches
322y Product Length/1st Dimension, in feet
323y Product Length/1st Dimension, in yards
324y Product Width/Diameter/2nd Dimension, in inches
325y Product Width/Diameter/2nd Dimension, in feet
326y Product Width/Diameter/2nd Dimension, in yards
327y Product Depth/Thickness/3rd Dimension, in inches
328y Product Depth/Thickness/3rd Dimension, in feet
329y Product Depth/Thickness/3rd Dimension, in yards
330y Container Gross Weight (kg)
331y Container Length/1st Dimension (Meters)
332y Container Width/Diameter/2nd Dimension (Meters)
```

```
333y Container Depth/Thickness/3rd Dimension (Meters)
334y Container Area (Square Meters)
335y Container Gross Volume (Liters)
336y Container Gross Volume (Cubic Meters)
340y Container Gross Weight (Pounds)
341y Container Length/1st Dimension, in inches
342y Container Length/1st Dimension, in feet
343y Container Length/1st Dimension in, in yards
344y Container Width/Diameter/2nd Dimension, in inches
345y Container Width/Diameter/2nd Dimension, in feet
346y Container Width/Diameter/2nd Dimension, in yards
347y Container Depth/Thickness/Height/3rd Dimension, in inches
348y Container Depth/Thickness/Height/3rd Dimension, in feet
349y Container Depth/Thickness/Height/3rd Dimension, in yards
350y Product Area (Square Inches)
351y Product Area (Square Feet)
352y Product Area (Square Yards)
353y Container Area (Square Inches)
354y Container Area (Square Feet)
355y Container Area (Suqare Yards)
356y Net Weight (Troy Ounces)
360y Product Volume (Quarts)
361y Product Volume (Gallons)
362y Container Gross Volume (Quarts)
363y Container Gross Volume (Gallons)
364y Product Volume (Cubic Inches)
365y Product Volume (Cubic Feet)
366y Product Volume (Cubic Yards)
367y Container Gross Volume (Cubic Inches)
368y Container Gross Volume (Cubic Feet)
369y Container Gross Volume (Cubic Yards)
37 Number of Units Contained
400 Customer Purchase Order Number
410 Ship To/Deliver To Location Code (Global Location Number)
411 Bill To/Invoice Location Code (Global Location Number)
412 Purchase From Location Code (Global Location Number)
420 Ship To/Deliver To Postal Code (Single Postal Authority)
421 Ship To/Deliver To Postal Code (Multiple Postal Authority)
8001 Roll Products - Width/Length/Core Diameter
8002 Electronic Serial Number (ESN) for Cellular Phone
8003 Global Returnable Asset Identifier
8004 Global Individual Asset Identifier
8005 Price per Unit of Measure
8100 Coupon Extended Code: Number System and Offer
8101 Coupon Extended Code: Number System, Offer, End of Offer
8102 Coupon Extended Code: Number System preceded by 090 Mutually Agreed Between
Trading Partners
91 Internal Company Codes
92 Internal Company Codes
93 Internal Company Codes
94 Internal Company Codes
95 Internal Company Codes
96 Internal Company Codes
97 Internal Company Codes
98 Internal Company Codes
99 Internal Company Codes
```

C.2 Fixed Length Fields

The GS1 Specification requires that some of the data to be encoded fits a standard length field. Zint will generate an error if the correct data lengths are not entered. The following table details which

AIs have fixed length data fields and how long the data should be for each:

Application Identifier	Number of characters (AI and Data Field)
00	20
01	16
02	16
03	16
04	18
11	8
12	8
13	8
14	8
15	8
16	8
17	8
18	8
19	8
20	4
31	10
32	10
33	10
34	10
35	10
36	10
41	16

[1] This information taken from Wikipedia and used under the terms of the GNU Free Documentation License.