Zint Barcode Generator and Zint Barcode Studio User Manual

5. Using the API

Zint has been written using the C language and currently only has an API for use with C language programs.

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 (or a GIF file called out.gif if libpng is not present) 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 data should be Unicode (UTF-8) 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 raster 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 text 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 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 raster image. The pixel 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 implemented by the external application:

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

5.5 Setting Options

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

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	Set various output file parameters	(none)

		(see section 5.8). [2]	
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, .bmp, .gif, .eps, .pcx, .svg or .txt	“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)	UNICODE_MODE
eci	integer	Extended Channel Interpretation mode	3
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

show_hrt	integer	Set to 0 to hide text.	1
dot_size	float	Size of dot used in dotty mode.	4.0/5.0
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 unsigned 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
ZINT_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.
ZINT_WARN_USES_ECI	Zint has automatically inserted an ECI character. The symbol may not be readable with some readers.
ZINT_ERROR_TOO_LONG	The input data is too long or too short for the selected symbology. No symbol has been generated.
ZINT_ERROR_INVALID_DATA	The data to be encoded includes characters which are not permitted by the selected symbology (e.g. alphabetic characters in an EAN symbol). No symbol has been generated.
ZINT_ERROR_INVALID_CHECK	An ISBN with an incorrect check digit has been entered. No symbol has been generated.
ZINT_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.
ZINT_ERROR_ENCODING_PROBLEM	A problem has occurred during encoding of the data. This should never happen. Please contact the developer if you encounter this error.
ZINT_ERROR_FILE_ACCESS	Zint was unable to open the requested output file. This is usually a file permissions problem.
ZINT_ERROR_MEMORY	Zint ran out of memory. This should only be a problem with legacy systems.

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	Barcode Name
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
14	BARCODE_EANX_CHK	EAN + Check Digit
16	BARCODE_EAN128	GS1-128 (UCC.EAN-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 Extended
32	BARCODE_TELEPEN	Telepen Alpha
34	BARCODE_UPCA	UPC A
35	BARCODE_UPCA_CHK	UPC A + Check Digit
37	BARCODE_UPCE	UPC E
38	BARCODE_UPCE_CHK	UPC E + Check Digit
40	BARCODE_POSTNET	PostNet
47	BARCODE_MSI_PLESSEY	MSI Plessey
49	BARCODE_FIM	FIM
50	BARCODE_LOGMARS	LOGMARS
51	BARCODE_PHARMA	Pharmacode One-Track
52	BARCODE_PZN	PZN
53	BARCODE_PHARMA_TWO	Pharmacode Two-Track
55	BARCODE_PDF417	PDF417
56	BARCODE_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 ECC200
72	BARCODE_EAN14	EAN-14
73	BARCODE_VIN	Vehincle Identification Number (America)
74	BARCODE_CODABLOCKF	Codablock-F
75	BARCODE_NVE18	NVE-18
76	BARCODE_JAPANPOST	Japanese Postal Code

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 ECC200
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
115	BARCODE_DOTCODE	DotCode
116	BARCODE_HANXIN	Han Xin (Chinese Sensible) Code
121	BARCODE_MAILMARK	Royal Mail 4-State Mailmark
128	BARCODE_AZRUNE	Aztec Runes
129	BARCODE_CODE32	Code 32
130	BARCODE_EANX_CC	Composite Symbol with EAN linear component
131	BARCODE_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
143	BARCODE_UPNQR	UPNQR (Univerzalni Plačilni Nalog QR)
144	BARCODE_ULTRA	Ultracode
145	BARCODE_RMQR	Rectangular Micro QR Code (rMQR)

5.8 Adjusting other Output Options

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

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

Value	Effect
0	No options selected.
BARCODE_BIND	Boundary bars above and below the symbol and between rows if stacking multiple symbols. [2]
BARCODE_BOX	Add a box surrounding the symbol and whitespace. [2]
BARCODE_STDOUT	Output the file to stdout.
READER_INIT	Add a reader initialisation symbol to the data before encoding.
SMALL_TEXT	Use a smaller font for the human readable text.
BOLD_TEXT	Embolden the human readable text.

CMYK_COLOUR	Select the CMYK colour space option for encapsulated PostScript files.
BARCODE_DOTTY_MODE	Plot a matrix symbol using dots rather than squares.
GS1_GS_SEPARATOR	Use GS instead FNC1 as GS1 separator.

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.
ESCAPE_MODE	Process input data for escape sequences.

DATA_MODE, UNICODE_MODE and GS1_MODE are mutually exclusive, whereas ESCAPE_MODE is optional. So, for example, you can set

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

whereas

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

is not valid. Permissible escape sequences are listed in section 4.1.

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, rMQR, GS1 DataBar-14 Stacked, PDF417 and MicroPDF417 - all of which have a fixed height.

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