1D (Linear) Barcode Decoding SDK (Enterprise Edition)

User's Guide

Contents

1. Introduction.

1.1 Scope

SDK is notated as **LC_EP_v.xx.xx**. Decoding library itself is notated as **LC_PRO.dll**.

Library interface is the same for Windows (XP…10), Linux, and certain embedded platforms. Both static and dynamic libraries are available.

Supported symbologies are separated into 3 groups as follows:

1. **EAN 13 (w/Add-on), EAN 8, UPCA, UPCE (w/Add-on), Code 39, Code 93, Code 128, Interleaved 2 of 5, Codabar**
2. **GS1 Databar:**
   - **GS1 Databar Omnidirectional, Stacked Omnidirectional**
   - **GS1 Databar Expanded, Expanded Stacked, Expanded Stacked CC-A and CC-B Composite**
   - **GS1 Databar Truncated, Limited, Stacked**
3. **PharmaCode, USPS PostNet, USPS IMB, New Zealand PostCode (NZP)**

The library allows to auto-discriminate symbologies within the first 2 groups. Symbologies from the third group must be set manually in advance to be decoded.

Symbol quality assessment is provided in accordance with ISO/IEC 15416 standard (first 2 groups).

Library processes **8-bit** Grayscale images only.

1.2 Normative references

ISO/IEC 15420:2009 - EAN/UPC bar code symbology specification
ISO/IEC 16388:2007 - Code 39 bar code symbology specification
ISO/IEC 15417:2007 - Code 128 bar code symbology specification
ISO/IEC 16390:2007 - Interleaved 2 of 5 bar code symbology specification
ISO/IEC 24724:2006 - Reduced Space Symbology (RSS) barcode symbology specification
ISO/IEC 15416:2000 - Bar code print quality test specification — Linear symbols
Laetus Pharmacode Guide, 4th and 5th Editions
USPS-STD-11 - Intelligent Mail Barcode (4-State Customer Barcode)
GS1 General Specifications, Version 12, Issue 1, Jan-2012

1.3 SDK composition

Decoding SDK contains:

- Windows DLL (**LC_PRO.DLL**) designed to perform 1D barcode search, recognition and decoding (written using C++).
- C# (**…/Sharp_LC_Pro.exe**) and C++ (**…/Demo_LC_Pro.exe**) Demo programs (source code included) to illustrate the DLL usage.
- Current User's Guide.

1.4 Features Description

Enterprise Edition specific Library features are described in the Table below:

Features Description	
CodeType	optional "Code Type" selection to speed up its recognition (allows to auto-discriminate symbologies within the first 2 groups). CodeType also can be equal to any code type or disjunction of code masks within first or second group.
Barcode Color	"Black-on-White" or "White-on-Black" symbol color settings
Multi Code Mode	decoding up to 10 barcodes in one image via variable setting
Quality Parameters	Quality Parameters assessment in accordance with ISO 15416
Decode Mode	includes setting for decoding bar codes having Quiet Zone violations
Checksum I25	optional checksum verification for "Interleaved 2 of 5" barcode
Checksum Codabar	optional checksum verification for "Codabar" barcode
Checksum C39	optional checksum verification for "Code 39" barcode
Time-out	setting for optional decode time limit
ScanDir & ScanStep	setting scan direction and distance between the scanlines
PharmacodeDir	setting scan direction for Pharmacode
IMB_EC	setting maximum number of errors to be corrected in USPS IMB barcode

1.5 Program session

Typical program session looks as follows:

```
Step 1. Connect decoder
Step 2. Create and set decoder options
Loop
        Step 3. Capture/read bitmap image
        Step 4. Process image
        Step 5. Request image and symbols info
        … // further application-specific data processing and interaction with user
End Loop
Step 6. Delete decoder options
Step 7. Disconnect decoder.
```

2. The Basic Interface Structures

The library includes the following structures:

struct TL_OptMode	the set of decoder options
struct TL_ImageInfo	features of decoded image
struct TL_Info	features of decoded symbols
struct LINEAR_Quality	Quality Parameters of decoded symbols
struct TIMB_L_QUALITY	Additional symbol Quality Parameters for IMB codes

2.1 Decoder options.

Decoder options are described in the Table below.

```
// decoder option modes
struct TL_OptMode
{
```

`int maxLcount;`	`/*Maximum number of barcodes. Possible values - in the range from 1 to 100, inclusive. It will be set internally as 1. Option realized for all types of code, except Pharmacode, PostNet and USPS IMB. For these codes it should be set to 1.`
`int typeCode;`	`/* Suppositive code type. Default value: TC_ANY (=-1), meaning that the search will be conducted within the First group of linear barcodes (except of Pharma, PostNet, USPS IMB and all RSS codes). See TYPE_CODE for mode details.*/`
`int timeOut;`	`/*TimeOut in milliseconds.`

	`Timeout = 0 - the infinite timeout.*/`

`int paramQ;`	`/*Definition of parameters of quality.`
	`1 - on, 0 – off*/`

`int checkSum_I25;`	`/*Definition of the control sum in Interleaved 2 of 5`
	`1 - on, 0 – off*/`

`int checkSum_C39;`	`/*Definition of the control sum in Code39`
	`1 - on, 0 – off*/`

`int quietZone;`	**`For compatibility with the legacy versions only.`**
	`Default value = -1 - QZ is set per barcode standard for each given symbology (normally 10X)`
	`Values: 4<=quietzone<=18 – manual setting if a priori information about quiet zone size is available;`
	`Values: 0<=quietzone<=3 are not allowed;`
	`automatically converts to Default value=-1, if chosen */`

`int smartmode;`	**`For compatibility with the legacy versions only.`**
	`Default value = 0.`
	`Is equal to number of couples bar+space at both sides of a barcode that we can treat as noise.`
	`Negative value means the unlimited quantity of "excessive" bars and spaces.`
	`Option is valid for all code types except Pharma, PostNet and USPS IMB.*/`

`int EC_Factor;`	`/*Maximum errors to be corrected in USPS IMB. 0<= opt_ECfactor <=4  (1..2 are preferable)*/`
`int pharmacodedir;`	`/*Direction of a pharma code decoding.*/`

`int scandir;`	`/*Scan directions. Option is valid for all code types except of PostNet and USPS IMB.`
	`See L_SCAN_DIR for more detail. */`

`int scanstep;`	`/*By Default is equal to 8.`
	`The value should conform to 1<= scanstep<=20. Option is valid for all code types except of PostNet and USPS IMB*/`

`int checksum_CB;`	`// Checksum in Codabar: 1 - on, 0 – off`
`int FullASCII_C39;`	`// Full ASCII mode in Code39 available`
`int colbeg;`	`// Left column of ROI. -1 for leftmost`
`int colend;`	`// Right column of ROI. -1 for rightmost`
`int rowbeg;`	`// Top row of  ROI. -1 for upper row`
`int rowend;`	`// Bottom row of ROI. -1 for lower row`
`int TADF_num;`	`// Number of 4-state bars in TADF Code`
`int color;`	`// Bar Color (1-Black  | 2-White | 3-Default)`

`int mode;`	`There are 3 decode modes:`
	`L_M_STANDARD - fully complies with the decoder version 19.06.7. Set as default.`
	`L_M_SMALL_QZ - decode symbols with a small quiet zone (less then 1X).`
	`L_M_STANDARD_AND_SMALL_QZ - combination of the two previous modes.`

```
};

The default option values are as following:

const TL_OptMode DefaultOptMode =
{
    1 // maxLcount
```

```
  ,-1 // typecode
  , 0 // timeout
  , 1 // paramq
  , 0 // checksum_I25
  , 0 // checksum_C39
  ,-1 // quietzone
  , 0 // smartmode
  , 2 // EC_Factor
  , 0 // pharmacodedir
  , 2 // scandir
  , 8 // scanstep
  , 0 // checksum_CB
  , 0 // Full ASCII code39
  ,-1 // colbeg
  ,-1 // colend
  ,-1 // rowbeg
  ,-1 // rowend
  ,65 // count of bars in TADF codes
  , 3 // default color
};
```

2.2 Image info.

```
// results of decoding the whole Image
struct TL_ImageInfo
{
int LCount;                 // number of well decoded symbols within image
int RejectionReason;        /* not L_SUCCESSFUL if no one symbol has been well
                            decoded.
                            See L_REJECTION_REASON for mode detail.*/
int BreakReason;            /* 0 - normal termination, 1 - termination by time-
                            out.
                            See L_BREAK_REASON for more detail.*/
};
```

2.3 Symbol Quality info.

Each decoded symbol is described by the following structures:

```
// Linear codes ISO 15416 Standard Quality Parameters.
struct LINEAR_QUALITY
{
    float decode;
    float symbol_contrast;
    float min_reflectance;
    float max_reflectance;
    float global_threshold;
    float min_edge_contrast;
    float modulation;
    float defects;
    float decodability;
    float decode_grad;
    float symbol_contrast_grad;
    float min_reflectance_grad;
    float global_threshold_grad;
    float min_edge_contrast_grad;
    float modulation_grad;
    float defects_grad;
    float decodability_grad;
    float overall_grade_grad;

    float decode_grad_av;
    float symbol_contrast_grad_av;
    float min_reflectance_grad_av;
    float global_threshold_grad_av;
    float min_edge_contrast_grad_av;
    float modulation_grad_av;
    float defects_grad_av;
    float decodability_grad_av;
    float overall_grade_grad_av;
};
```

Important!
The overall symbol grade calculated as the arithmetic mean of the scan reflectance profile grades using the decoded scan lines only.

2.4 4 State (IMB, NZP) Quality Parameters

```
/// Additional symbol Quality Parameters for 4 State codes
const int bar_limit = 100;

struct TIMB_L_QUALITY
{
intSize_index_T;                      // Length of array TrackerHeight
intSize_index_A;                      // Length of array AscenderHeight
intSize_index_D;                      // Length of array DescenderHeight
intSize_index_F;                      // Length of array  FullHeight
float BarWidth [bar_limit];
float SpaceWidth [bar_limit];
float BarPitch [bar_limit];
float BarHeight [bar_limit];
float TrackerHeight [bar_limit];      // not 0 if BarType[i] = 'T'
float AscenderHeight [bar_limit];     // not 0 if BarType[i] = 'A'
float DescenderHeight [bar_limit];    // not 0 if BarType[i] = 'D'
float FullHeight [bar_limit];         // not 0 if BarType[i] = 'F'
float BarRotation [bar_limit];
char  BarType [bar_limit];            // 'T' or 'A' or 'D' or 'F'
intBarCol [bar_limit];                // column where base line intersects bar
intBarRow [bar_limit];                // row where base line intersects bar
float BarReflectance[bar_limit];
float SpaceReflectance[bar_limit];
float
BackgroundReflectance[bar_limit];
float PrintReflectanceDifference
[bar_limit];
float BaselineShift [bar_limit];      // the bar shift in vertical direction
                                      from estimated base (middle) line
intBarInk [bar_limit];                // the average over-ink size of the bars
intVoidGrade[bar_limit];              // the average void spacing (area
                                      without ink) of the bars
intOverinkGrade [bar_limit];
intminClearZoneLeft;
intminClearZoneRight;
intminClearZoneTop;
intminClearZoneBottom;
intgradeClearZoneLeft;
intgradeClearZoneRight;
intgradeClearZoneTop;
intgradeClearZoneBottom;
int   actTADFnum;                     Actual number of bars in TADF code <=99
 };
```

2.5 Scanline descriptor

```
// scan line descriptor.
struct  LINEAR_SCAN_LINE
{
short int scan_line_start_x;         // column of scan line begin
short int scan_line_start_y;         // row of scan line begin
short int scan_line_end_x;           // column of scan line end
short int scan_line_end_y;           // row of scan line end
bool  first;                         // flag of main scan line
bool  quietzone;                     // Quiet Zone OK
bool  checksum;                      // check sum OK
unsigned char length;                // number of decoded characters
char* dataRes;                       // result of decoding the scan line
};
// Descriptor of 10 scan lines in each L_Info
typedef  LINEAR_SCAN_LINE  SCAN_LINE_ALL[10];
```

2.6 Linear symbol info

```
// Linear symbol properties (per each decoded symbol)
struct  TL_Info{
short int     RejectionReason;   /* Rejection qualifier.
                                 Available Rejection Reasons.
                                 0 - Successful.
                                 6 - The Checksum failed.
                                 7 - Code not Confirmed on several scan lines.
                                 8 - No Quiet Zone to the left or to the right
                                 of the barcode.
                                 9 - obligatory 2D-symbol is not found
                                 See L_REJECTION_REASON for more detail.*/
short int       rowcols[8];      // Coordinates of barcode rectangle
short int       type;            /*Actual type of a barcode.
                                 See TYPE_CODE for more detail.*/
unsigned char   idx_scan_line;   /* Index of "main" scan line
```

```
                                  (0<=idx_scan_line<10)*/
unsigned char   pchlen;          // Length of decoded byte array.
char*           pch;             // Pointer to decoded byte array.
SCAN_LINE_ALL   sl;              // Descriptors of 10 scan lines
LINEAR_QUALITY  lq;              // ISO 15416 Quality Parameters
TIMB_L_QUALITY* add_lq;          // not null if code type is IMB
};
```

2.7 The Constants.

```
enum TYPE_CODE{
    TC_ANY                  = -1,
    TC_TCODE128             = 1,
    TC_TCODE39              = 2,
    TC_TCODABAR             = 3,
    TC_TINTERLEAVED         = 4,
    TC_TEAN13               = 5,
    TC_TEAN8                = 6,
    TC_TUPCE                = 7,

    TC_TPOSTNET             = 8,
    TC_TIMB                 = 9,  //!< USPS IMB
    TC_TPHARMACODE          = 10,

    TC_TRSS                 = 11, //!< RSS-14 (Stacked and/or Composite)
    TC_TRSS_L               = 12, //!< RSS Limited (Composite)
    TC_TRSS_E               = 13, //!< RSS Expanded (Stacked and/or Composite)

    TC_TEAN13EX             = 14,
    TC_TUPCEEX              = 15,
    TC_TCODE93              = 16,

    TC_TADF                 = 17, //!< Contains 4-state bars from 10 to 99
    TC_TNZP                 = 18, //!< New Zeland Post Code

    TCM_TCODE128            = 0x00200,
    TCM_TCODE39             = 0x00400,
    TCM_TCODABAR            = 0x00800,
    TCM_TINTERLEAVED        = 0x01000,
    TCM_TEAN13              = 0x02000,
    TCM_TEAN8               = 0x04000,
    TCM_TUPCE               = 0x08000,
    TCM_TEAN13EX            = 0x400000, // 0x100<<14
    TCM_TUPCEEX             = 0x800000,

    TCM_ANY                 = 0x0FE00,

    TCM_TRSS                = 0x080000,
    TCM_TRSS_L              = 0x100000,
    TCM_TRSS_E              = 0x200000,

    TCM_TRSS_ALL            = 0x380000
};
```

There are few options available when choosing TYPE_CODE constant, as follows:

1. **TC_T*** ("*" – wildcard ranging from 1 to 18) – sets a certain symbology to be decoded (when the prior information about the type of the code within the image is available)

2. If there are few known codes within the image one can set a few constants **TC_T*** at once to expedite the barcode recognition process, for example:

   ```
   Opt1.typecode = TCM_TCODE128 | TCM_TCODABAR | TCM_TEAN8 (Group 1)
   Or
   Opt1.typecode = TCM_ TRSS | TCM_ TRSS_L (Group 2);
   ```

Important:
Group 1&2 codes shall not be mixed together when specifying "typecode" option.

3. If no prior information about the type of the codes within the image is available,
   - **TC_T*** should be set to "-1" – default setting - (for Group 1 symbologies) or
   - "**TCM_TRSS_ALL**" (for Group 2 symbologies).

4. PharmaCode, PostNet, USPS IMB, and NZP can be searched/decoded ONLY when using setting TypeCode=TC_TPHARMACODE or TypeCode=TC_TPOSTNET or ….

```
enum L_REJECTION_REASON{
    L_SUCCESSFUL          = 0,
    L_NO_LINEAR_FOUND     = 1,
    L_POOR_IMAGE_QUALITY  = 2,
    L_RR_UNKNOWN          = 3,
    L_RS_ERROR            = 5,
    L_CHECKSUM_FAILED     = 6,
    L_NOT_CONFIRMED       = 7,
    L_NOT_QUIET_ZONE      = 8,

    L_2D_FAILED          = 9 //!< obligatory 2D-symbol is not found (for RSS
Composite)
};

enum L_BREAK_REASON{
    L_ALL_INSPECTED       = 0, //!< Normal termination (whole image was
inspected)
    L_TIMEOUT             = 1, //!< Termination by time-out
    L_TERMINATED          = 2  //!< Termination by user break (not
implemented)
};

/* Directions of scan */
enum L_SCAN_DIR{
    L_SCAN_VERTICAL       = 0,
    L_SCAN_HORIZONTAL     = 1,
    L_SCAN_ALL            = 2
```

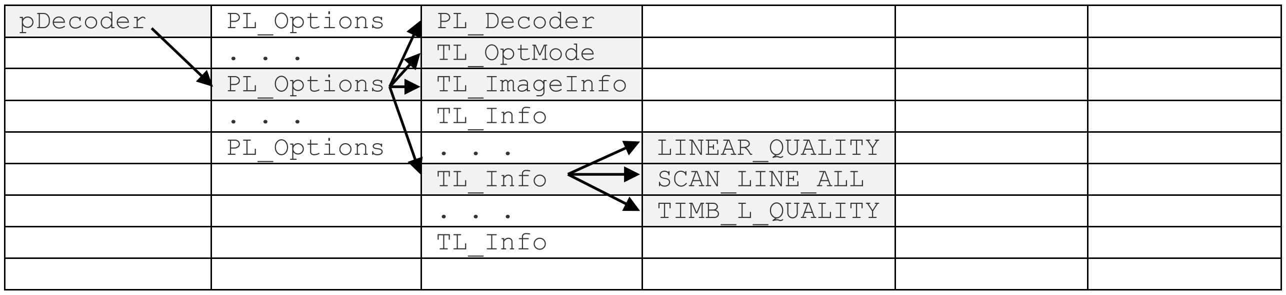
```
};

/* Directions of Pharmacode*/
enum L_PHARMACODEDIR{
    L_LR   = 0, //!< Barcode goes approximately from image left to image
right
    L_TB   = 1, //!< From image top to bottom
    L_RL   = 2, //!< From image right to left
    L_BT   = 3  //!< From image bottom to top
};
```

2.8 Type definitions.

```
typedef unsigned char*  TRow;           // pointer to image rows
typedef void*           PL_Decoder;     // handle of L_Decoder
typedef void*           PL_Options;     // handle of L_Decoder Options
typedef TL_ImageInfo*   PL_ImageInfo;   // descriptor of structure ImageInfo
typedef TL_Info*        PL_Info;        // descriptor of structure Linear
                                        Symbol Info
typedef LINEAR_QUALITY* PL_Quality;     // descriptor of Quality Parameter
typedef SCAN_LINE_ALL*  PL_Scanlines;   // pointer to array of 10 scan lines
typedef TIMB_L_QUALITY* PL_IMBQuality;  // pointer to IMB Quality Parameters

Following figure represents the typical links during application execution.
```



3. The Interface Procedures and Functions

The interface objects are defined like following.
Description of the interface procedures is below.

3.1 Connect_L_Decoder

PL_Decoder Connect_L_Decoder (int maxrowcount, int maxcolcount);
Description.
Function generates new instance of class encapsulating the decoder functionality.

Parameters.

Maximums of horizontal and vertical image sizes.

Maximum image dimensions are 4000x4000 in Windows and 844x640 in iVu Embedded platform. The identical results appear if we pass identical dimensions into Connect_L_Decoder in different environments (for instance **maxrowcount=640, maxcolcount=844**).

Return value.

Pointer to decoder in success, or NULL otherwise.

3.2 Disconnect_L_Decoder

void Disconnect_L_Decoder(PL_Decoder &);

Description.

Procedure destroys decoder class and frees memory.

Parameter.

Pointer to decoder. Decoder should be connected.

3.3 Create_L_Options

Class TL_Options encapsulates the decoder options and methods of image processing and inspection.

PL_Options Create_L_Options (PL_Decoder ,TL_OptMode);

Description.

Function generates new class to decode image with certain options.

Parameters.

- Pointer to decoder.
- Pointer to option modes that specify the way of image processing

Return value.

The handler that provides decoding of the image with desirable options.
You can store this handler and come back to it in any decoding cycle.

3.4 Delete_ L_Options

void Delete_L_Options (PL_Options &);

Description.

The function destroys a handler.

Parameters.

- Handler of decoder.

3.5 Decode_L_Bits

```
int     Decode_L_Bits    (           PL_Options     popt,
                         int         actualrowcount,
                         int         actualcolcount,
                         TRow*       prows);
```

Description.

The function processes an image and fills ImageInfo and array of SymbolInfo's.

Parameters.

- Handler produced by 3.3
- Number of image rows
- Number of image columns
- Array of pointers to image rows. Every row is a byte array with 8-bit pixel intensities.

(We have **typedef unsigned char* TRow**;)

Return value.

0 if no one symbol was decoded, >0 otherwise.

3.6 GetL_ImageInfo

```
// inspects image info after decoding
PL_ImageInfo GetL_ImageInfo    (PL_Options);
```

Description.

The function returns image info.

Return value.

Pointer to TL_Image Info.

3.7 GetL_Info

```
PL_Info    GetL_Info        (PL_Options ,int LNum);
```

Description.

The function returns symbol info.

Parameters.

- Handler of decoder
- Number (index) of decoded symbol in image.

If no symbols were decoded we return Info about the most probable symbol location.

Return value.

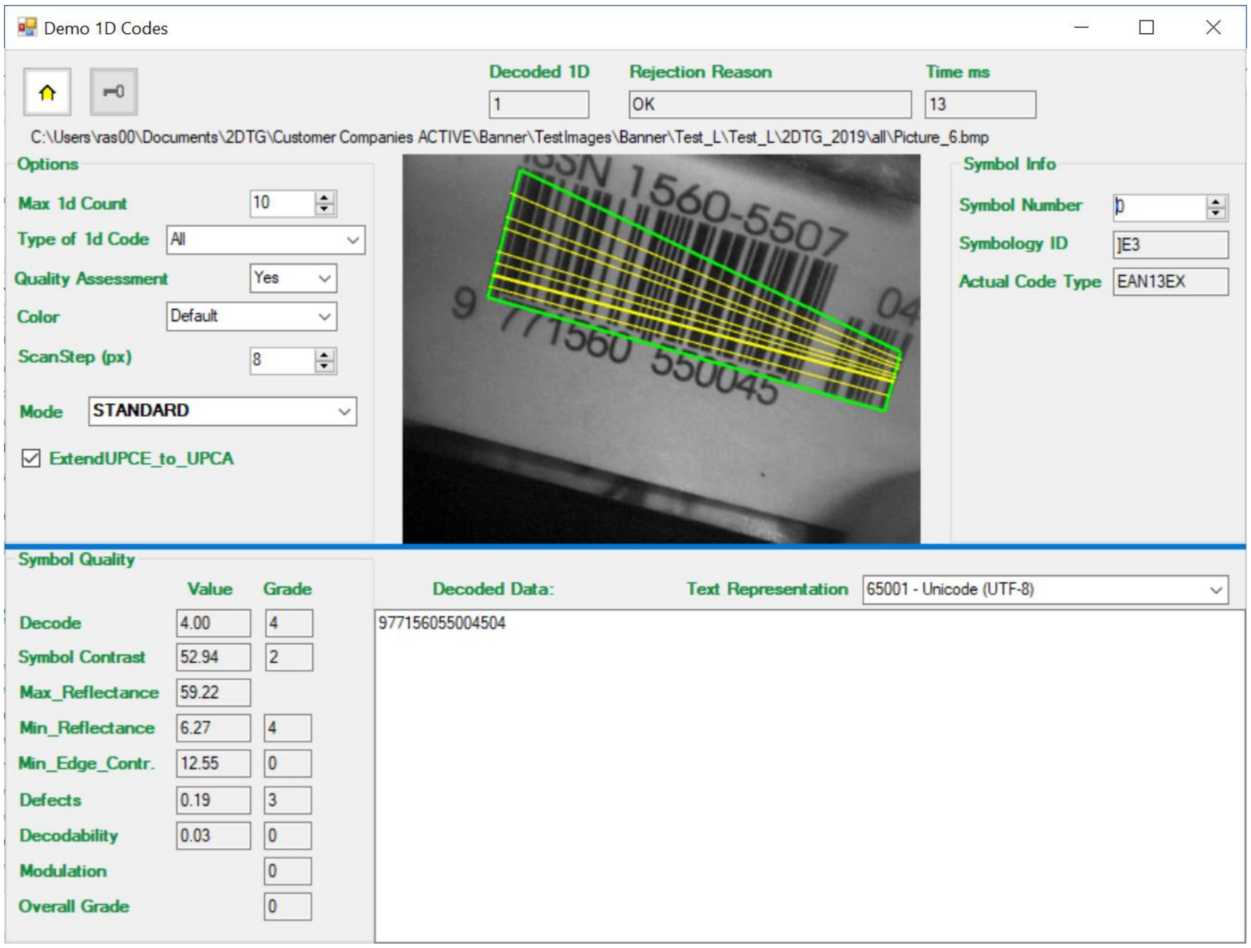
Pointer to TL_SymbolInfo or NULL, if the LNum is out of range..

4. Demo applications and Decoding tips

Decoding Library comes with the Demo applications build in C# and C++ in MSVC development environment.

4.1 C# Demo application – GUI

GUI illustrates all major features of the Library:



Decoded 1D: 1

Rejection Reason: OK

Time ms: 13

C:\Users\ras00\Documents\2DTG\Customer Companies ACTIVE\Banner\TestImages\Banner\Test_L\Test_L\2DTG_2019\all\Picture_6.bmp

Options

Max 1d Count	10
Type of 1d Code	All
Quality Assessment	Yes
Color	Default
ScanStep (px)	8
Mode	STANDARD

☑ ExtendUPCE_to_UPCA

Symbol Info

Symbol Number	0
Symbology ID	]E3
Actual Code Type	EAN13EX

Symbol Quality

	Value	Grade
Decode	4.00	4
Symbol Contrast	52.94	2
Max_Reflectance	59.22	
Min_Reflectance	6.27	4
Min_Edge_Contr.	12.55	0
Defects	0.19	3
Decodability	0.03	0
Modulation		0
Overall Grade		0

Decoded Data:

Text Representation: 65001 - Unicode (UTF-8)

```
977156055004504
```

Decode Settings Options (described also in the Section 1.4):

- **Max 1D count** – number of barcode symbols within an image (if known in advance). Allowed range (1 - 100). Default value – “10”. Exceptions: Pharmacode, PostNet, USPS IMB and New Zealand postal codes – this parameter must be set to “1”.
- **Type of 1D Code**:
  - **ALL** - search within all linear barcode types except GS1 Databar/RSS Family, Pharma Code and postal codes;
  - **RSS All** - search within GS1 Databar/RSS Family (recommended setting):
    - **RSS** – search within GS1 Databar Omnidirectional, Stacked Omnidirectional
    - **RSS-E** – search within GS1 Databar Expanded, Expanded Stacked, Expanded Stacked CC-A and CC-B Composite
    - **RSS-L** – search within GS1 Databar Truncated, Limited, Stacked
  - Individual symbology setting: **Pharma**, **PostNet**, **USPS IMB, New Zealand**
- **Quality Assessment** - setting Quality Parameters assessment in accordance with ISO 15416 - – YES (default)/NO
- **Color** – Black-on-White, White-on-Black, Default (any color) - symbol color settings for reading images printed in fluorescent ink (applicable to postal symbologies only)
- **ScanStep** (px) – allows to set a distance between the scanlines. Allowed range (1 - 20) - default – 8 pixels.
- **Mode** – three Decode modes as follows:
  - **L_M_STANDARD** – applicable to all barcode symbologies in 2DTG offering (default setting).
  - **L_M_SMALL_QZ** - decodes symbols with a small Quiet Zone (less then 1X). Supports:
    - **EAN 13 (w/Add-on), EAN 8, UPCA, Code 39, Code 93, Code 128**, BUT NOT:
    - **Interleaved 2/5, UPCE (w/Add-on) and CODABAR** symbols - full quiet zone is still required.
  - **L_M_STANDARD_AND_SMALL_QZ** - combination of the two previous modes.
- **ExtendUPCE_to UPCA** - “UPC-E to UPC-A” conversion function. Default - “ON”.

Overall decode info:

- **Decoded 1D** – number of barcodes decoded in this image
- **Rejection Reason** – returns decode result:
  - **“OK” (0)** successful decoding or:
  - Error Code - in some cases decoding library can return certain error codes associated with the decoding process. They are as follows:
    - **No Linear Found (1)** – no “structured barcode-like formations” found within the image

    - **Poor Image Quality (2)** – bad image quality
    - **Reed-Solomon Error (5)** – too many Reed-Solomon errors
    - **Checksum Failed (6)** – Checksum vetting failed
    - **Not Confirmed (7)** – decode results not confirmed by decoding on different skylines
    - **Quiet Zone Error (8)** – Quiet Zone violation
- **Time (ms)** – total decode time (all barcodes within an image)

Symbol Info:

- **Symbol Number** – symbol for which the decode result is displayed (starts with number "0") assuming multiple number of symbols in the image
- **Symbology ID** – GS1/Regular barcode identifier for displayed symbol
- **Actual Code Type** – displays decoded code type

Symbol Quality – results of the symbol quality assessment in accordance with ISO/IEC 15416

4.2 Decoding tips and limitations for "Small QZ decoding" mode

L_M_SMALL_QZ mode proved to be every effective in decoding symbols with a small Quiet Zone (less then 1X) and "better than average" effective in decoding symbols with uneven illumination.

There are some limitations, however, that should be considered when using this decode mode:

- **L_M_SMALL_QZ** - decodes symbols with a small Quiet Zone (less then 1X) for the following symbologies:
    - **EAN 13 (w/Add-on), EAN 8, UPCA, Code 39, Code 93, Code 128**, BUT NOT for:
    - **Interleaved 2/5, UPCE (w/Add-on) and CODABAR** symbols - full quiet zone is still required.

- Multiple symbols decoding within 1 image is not supported.
- Quality parameters evaluation for this mode is not currently supported.
- 'White on black' symbols decoding for this mode is not currently supported.

Major advantages of this new decoding algorithm in comparison with the legacy algorithms are as follows:

- Small Quiet Zone issue is resolved for most widely used barcodes (it is allowed to be less than 1X now (standard requirement – 10X))

- Success decode rate in our stress tests (over 700 samples) increased by ~ 7%
- Decode speed for this mode (over the same samples) increased by ~ 30%
- Decode speed in "Combined mode" increased by ~ 15%

Recommendation: "Combined mode" setting should be used when "Multiple mode" is not required.

4.3 C++ Demo application - Source Code Description

Note: The secondary functions (such as different kinds of print) are represented in "ForRTest.cpp".

The application itself is represented in file "TestApp.cpp" :

```
#include "stdafx.h"

#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include "LoadBMP.cpp"
#include "..\\Source\\l_types.h"

#include "ForRTest.h"
#include "ForRTest.cpp"

//----------------------------------------------------------------------------

 char* dir1="Addon_Img\\";
 char* files[]={
 "im1.bmp"
,"pic-6.bmp"
,"ad21826.bmp"
,"ad86104.bmp"
,"ad45856.bmp"
,NULL   // last member of array must be NULL
 };
void MainTask_Fn(void)
{
  int result;   int  row;
  int rowcount, colcount;
  int maxR = 3000; int maxC = 3000;
  TRow*     pbits;
  TRow      pmembits;  // Image in Memory
  int ResLoadBMP;
  int i;

  int   countFalse =0;
  int   csumErr=0, confErr=0, qzErr=0, countOK=0;
  char  dir [100];
  char  path[100];
  PL_Decoder    pDecoder;
  PL_Options    pDecWithOpt;

  LINEAR_QUALITY*  lq;
  PL_ImageInfo  imageinfo;
  PL_Info       Linfo;
  int RR, BR, num;
  int OKCodes = 0;
  char* pch;
  char Symbology[4] = {0,0,0,0};

  printf("  Version date %s, time %s%\n\n", GetVersionDate_L(),
GetVersionTime_L());

  pDecoder = Connect_L_Decoder(maxR,maxC);
  if(pDecoder==NULL) return ;

  pmembits = (TRow) malloc(maxR*maxC);       //  Image in Memory
  pbits    = (TRow*) malloc(maxR*sizeof(TRow)); // pointers to ScanLines

  for (row = 0; row < maxR; row++){
      pbits[row] = &pmembits[maxC*row];
  }

  strcpy(dir,dir1);

// Setting the options of 1D decoder:
  TL_OptMode opt1 = { 10,  // int maxLcount;
                     -1,  // int typecode any;
                     // 4,  // interleaved
                     // 8,  // int typecode PN;
                     // 9,  // int typecode IMB;
                     // 10, // PHARMACODE

                      0,  // int timeout;
                      1,  // int paramq;
                      0,  // int checksum_I25;
                      0,  // int checksum_C39;
                     -1,  // int quietzone;
                       0,//1,//3, // int smartmode
                       0, // int EC_Factor
                       4, // int pharmacodedir (LRL)
                       2, //int scandir = vertical+horizontal;
                       8, //int scanstep;
                       0, // checksum_CB;
                       0, // int FullASCII_C39;
                      -1, // 250,  //int colbeg
                      -1, // 440,  //int colend      //200, 50, 440, 430
                      -1, //  50,  //int rowbeg
                      -1  // 430,  //int rowend
                      ,99 // TADF_num
                      ,3  // color
                     };
  pDecWithOpt = Create_L_Options (pDecoder ,opt1);

  long currTime1=0;

  printf("maxLcount=%d \n",opt1.maxLcount);
  printf("typecode=%d \n",opt1.typecode);
  printf("timeout=%d \n",opt1.timeout);
  printf("paramq=%d \n",opt1.paramq);
  printf("checksum_I25=%d \n",opt1.checksum_I25);
  printf("checksum_C39=%d \n",opt1.checksum_C39);
//  printf("checksum_CB=%d \n",opt1.checkSum_CB);
  printf("quietzone=%d \n",opt1.quietzone);
  printf("smartmode=%d \n",opt1.smartmode);
//  printf("FullASCII=%d \n",opt1.FullASCII_C39);
  printf("scandir=%d \n",opt1.scandir);
  printf("scanstep=%d \n",opt1.scanstep);
//  printf("TADF_num=%d \n",opt1.TADF_num);

  printf("dir=%s \n",dir1);

  int count = 0;
  double TotalTime = 0.0;

  for(int j=0;files[j]!=NULL; j++){
   strcpy(path,dir);
   strcat(path,files[j]);

   printf("\n");
   printf("%s ", files[j]);
   rowcount = maxR;
   colcount = maxC;

// opening the "bmp" file:
   ResLoadBMP = LoadBMP (path , pbits ,rowcount ,colcount);
// rowcount and colcount can change in this call

   if(ResLoadBMP != 0) {
     printf("ResLoadBMP == %d \n",ResLoadBMP);  continue;
   }//return 1;

   count++;

LARGE_INTEGER liFreq;
LARGE_INTEGER liStart;
LARGE_INTEGER liStop;
double m_DecodeTime;
::QueryPerformanceFrequency(&liFreq);
::QueryPerformanceCounter(&liStart); // Do your processing

// call main decoding function:
   result =  Decode_L_Bits(pDecWithOpt, rowcount,colcount,pbits);

::QueryPerformanceCounter(&liStop);
LONGLONG llTimeDiff = liStop.QuadPart - liStart.QuadPart;
m_DecodeTime=(double)llTimeDiff*1000/liFreq.QuadPart;
```

```
printf(",  time=%7.2f",m_DecodeTime);
TotalTime += m_DecodeTime;

   if(result<0){
     printf(" decoder == NULL!!!!!!"); return ;
   }

    print_LN();
    imageinfo = GetL_ImageInfo(pDecWithOpt);
    if (imageinfo==NULL)
      continue;
    RR = imageinfo->RejectionReason;
    BR = imageinfo->BreakReason;

      if(result>0)
    {
      num = imageinfo->LCount;

      for(int idx=0; idx<num; idx++) {
       if(idx>0) print_LN();
        Linfo = GetL_Info(pDecWithOpt,idx);
        RR     = Linfo->RejectionReason;
            OKCodes += (RR==L_SUCCESSFUL);
        //len    = Linfo->pchlen;
        pch    = Linfo->pch;

      print_TYPE_CODE_Result(Linfo->type,RR);

// setting Symbology ID
            Symbology[0] = pch[-3];
            Symbology[1] = pch[-2];
            Symbology[2] = pch[-1];
        print_s("  ");
        print_s(Symbology);
        print_s("  ");

// print decode result
        print_s(pch);
            print_LN();

        lq = &Linfo->lq;

// print quality parameters
        //refle   contr   edgec   modu    defec   decod   D O
        printLetterGrade(lq->min_reflectance_grad,   lq->min_reflectance);
        printLetterGrade(lq->symbol_contrast_grad,   lq->symbol_contrast);
        printLetterGrade(lq->min_edge_contrast_grad, lq->min_edge_contrast);
        printLetterGrade(lq->modulation_grad,        lq->modulation);
        printLetterGrade(lq->defects_grad,           lq->defects);
        printLetterGrade(lq->decodability_grad,      lq->decodability);
        printLetterGrade(lq->decode_grad);
        printLetterGrade(lq->overall_grade_grad);
      } //for idx
    }else{ //(result<0)
      if (BR == L_TIMEOUT) print_s("TimeOut"); else
      if (BR == L_TERMINATED) print_s("Terminated"); else
      print_s("Unknown");
      }//if(result>0)
      print_LN();
  } //!!!!
  printf("\n\n Total Files = %d", count);
  printf("\n OK Codes     = %d \n\n", OKCodes);
  if (count>0)
    printf(",  Avr.time = %8.2f\n",TotalTime/count);

  Delete_L_Options (pDecWithOpt);
  Disconnect_L_Decoder(pDecoder);

  free(pbits);
  free(pmembits);

  printf(">");
  char ch; scanf("%c",&ch);

  return;
}

int _tmain(int argc, _TCHAR* argv[])
{
      MainTask_Fn();
      return 0;
}
```

5. GS1 Compliance

GS1 System uses a special start combination - Symbology Identifier - to differentiate the GS1 symbol from the other symbols. It enables scanners to process the information according to the GS1 System Rules.

The symbology identifier is a three-character data string comprising a flag character, code character, and a modifier character.

Character	Description
]	The flag character (which has an ASCII value of 93). This denotes that the two characters following it are Symbol Identifier characters.
c	The code character. This denotes the type of symbology.
m	The modifier character. This indicates the mode in which the symbology is used.

Note: If used, the symbology identifier is transmitted as a prefix to the data message.

The symbology identifiers used in the GS1 System for the linear barcodes are shown below.

Symbology Identifier*	Symbology Format	Content
] E 0	EAN-13, UPC-A, or UPC-E	13 digits
] E 1	Two-digit Add-On Symbol	2 digits
] E 2	Five-digit Add-On Symbol	5 digits
] E 3	EAN-13, UPC-A, or UPC-E with Add-On Symbol**	15 or 18 digits
] E 4	EAN-8	8 digits
] I 1	ITF-14	14 digits
] C 1	GS1-128	Standard AI (Application Identifier) Element Strings
] e 0	GS1 DataBar	Standard AI Element Strings
] e 1	GS1 Composite	Data packet containing the data following an encoded symbol separator character.
] e 2	GS1 Composite	Data packet containing the data following an escape mechanism character.

Notes:

* Symbology identifiers are case sensitive.

** Bar codes with Add-On Symbols may be considered either as <two separate symbols, each of which is transmitted separately with its own symbology identifier (not implemented), or as a single data packet with Symbology ID =”]E3” (implemented).

According to ISO/IEC 15420:2009, Add-On symbol can be located only on the right side of the main barcode. Any bars on the left will be ignored by the decoding algorithm.

2DTG’s decoding library returns Symbology Identifier that can be used by GS1 users when building their applications. The symbology identifier is transmitted as a prefix to the data message at positions -3,-2,-1.

6. Licensing / Evaluation

Stand-alone license is locked to the computer, on which it was activated, and may not be transferred to another computer. If the computer was upgraded or rebuilt the license may still be valid if its major components had not been changed.

Important:

Licensing mechanism requires two additional files for unlock and operation (in addition to Decoding Library):

- **IP2Lib64.dll** or **IP2Lib32.dll;** and
- Configuration XML-file - **LC – Enterprise.xml**.
- Product LOGO file (**ProdLogo_**.bmp**) is also recommended but not strictly required.

By default, 2DTG supplies all these files located in the same folder as demo-application that would call the library.

We recommend activating decoding library by starting our Demo application and following the Activation Instructions below.

If you are planning to call decoding library from your own application, please, make sure to copy those 3 files to the folder where your application is located.