User's Guide

Contents

1. Introduction.

1.1 Scope

This document is applicable to the Aztec Code Decoding SDK.

SDK is notated as **AZ_v.xx.xx** and contains both 32 bit and 64-bit versions.

The Library interface is the same for Windows, Linux, and certain embedded platforms. Both static and dynamic libraries are available.

The library is designed to decode Aztec Codes in accordance with ISO/IEC 24778 Symbology specification.

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

1.2 Normative references

ISO/IEC 24778:2008 - Aztec Code bar code symbology specification

1.3 SDK package composition

Decoding SDK contains:

- C++ Windows DLL (**icDecLibAZ.DLL**) written in MSVS 2017 and designed to perform Aztec Code search, recognition and decoding.
- C++ Demo program **(…/Demo_AztecCpp .exe**) and C# Demo program **(…/DemoAztecSharp .exe**) built in MSVS development environment (both come with source code) - to illustrate the DLL usage.
- Current User's Guide.

1.4 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 TAZ_OptMode - the set of decoder options,
struct TAZ_ImageInfo - features of decoded image,
struct TAZ_Info - features of decoded symbols,
struct TAZ_Quality - Quality Parameters of decoded symbols (NOT currently available).

2.1 Decoder options

```
/// decoder option modes
struct TAZ_OptMode
{
  int maxAZCount;   //!< 1 by default (NOT currently available)
  int cellColor;    //! (NOT currently available)
  int speedMode;    //!< always SP_ROBUST (NOT currently available)
  int qualityMask;  //!< DM_QM_NO by default (NOT currently available)
  int labelMode;    //!< always LM_NORMAL (NOT currently available)
  int timeOut;      //!< timeOut in mls. (NOT currently available)
  int filterMode;   //!< FM_NON by default (NOT currently available)
  int qzMode;
};
```

2.2 Image info

```
/// results of decoding the whole Image
struct TAZ_ImageInfo
{
  int AZCount;        //!< number of well decoded symbols within image
  int RejectionReason;//!< not 0 if no one code has been well decoded
  int BreakReason;    //!< 0 - normal termination, 1 - termination by time-
out
 };
```

ImageInfo.AZCount = 1 if any Rectangle-shaped object was detected in image.

2.3 Symbol info

Each decoded symbol is described by the following structures:

```
/// Aztec Quality Parameters
struct TAZ_Quality
{
  float sc;    //!< Symbol Contrast
  float an;    //!< Axial Non uniformity
  float gn;    //!< Grid  Non uniformity
  float uec;   //!< Unused Error Correction
```

```
  /// Grades:
  float scG;   //!< Symbol Contrast Grade
  float anG;   //!< Axial Non uniformity Grade
  float gnG;   //!< Grid  Non uniformity Grade
  float uecG;  //!< Unused Error Correction Grade
  float modG;  //!< Modulation Grade
  float fpdG;  //!< Fixed Pattern Damage Grade
  float deG;   //!< Decode Grade
  float scanG; //!< Scan Grade (minimum of Grades)
  float overall_grade;              //!< minimum of grades
};

/// result of decoding of Aztec code symbol in image

struct TAZ_Info
{
  float          rowcols[8];    //!< symbol corner coordinates
  int            pchlen;        //!< length of decoded byte array
  unsigned char* pch;           //!< pointer to that array
  int            RSErr;         //!< number of Reed Solomon errors
  int            Dim;           //!< dimensions of Aztec code
  int            Rune_AZS;      //!< 0 for AZC, 1 for compact, 2 for Rune
  int            Layers;
  int            MSGLen;
  TAZ_Quality    quality;       //!< symbol Quality Parameters
};
```

2.4 The Constants

```
enum AZ_DECODER_SPEED{
 AZ_SP_ROBUST          = 0
};

/// \enum QUALITY_MASK bits of mask:
enum AZ_QUALITY_MASK{
 AZ_QM_NO       = 0X0000,
 AZ_QM_ALL      = 0x7FFF
};

enum AZ_FILTER_MODE{
 FM_NON         = 0, //!< No filter
 FM_AUTO        = 1  //!< Digital auto-focus.
};

enum AZ_QZ_MODE{
 AZ_QZ_NORMAL   = 0 //!< allows QZ>= 3 module sizes, decoder in average more
fast and robust
,AZ_QZ_SMALL    = 1 //!< allows QZ>= 1 module size, affects speed and
robustness
};

enum AZ_BREAK_REASON{    //!< invalid anyware except of TI platform
//----------------------
 AZ_ALL_INSPECTED    = 0 //!< no breaks occurred
```

(NOT currently available)

```
,AZ_TIMEOUT          = 1 //!< termination by time out
,AZ_TERMINATED       = 2 //!< termination by user break
};
```

2.5 Type definitions

```
typedef void*           PAZ_Decoder;   //!< handler of Aztec Decoder
typedef void*           PAZ_Options;   //!< handler of Decoder Options
typedef TAZ_ImageInfo*  PAZ_ImageInfo; //!< pointer to Image Info
typedef TAZ_Quality*    PAZ_Quality;   //!< pointer to symbol Quality
typedef TAZ_Info*       PAZ_Info;      //!< pointer to symbol Info
typedef unsigned char*  TRow;          //!< pointer to bitmap line

/// The function creates Aztec Decoder and returns Decoder handler
PAZ_Decoder Connect_AZ_Decoder(int maxrow, int maxcol);

/// The function destroys Aztec Decoder
void        Disconnect_AZ_Decoder(PAZ_Decoder &pDecoder);

/// The function creates Decoder Options and returns Options handler
PDM_Options Create_AZ_Options(PAZ_Decoder pDecoder, TAZ_OptMode optmode);

/// The function destroys Decoder Options
void        Delete_AZ_Options(PAZ_Options &pOptions);

/// The function decodes array ppbits with given Options
int Decode_AZ_Bits(PAZ_Options pOptions, int rowcount, int colcount, TRow*
ppbits);

/// The function returnes the ImageInfo of last decoded Image
PAZ_ImageInfo Get_AZ_ImageInfo(PAZ_Options pOptions);

/// The function returnes the DM_Info(dmNum)
PAZ_Info      Get_AZ_Info(PAZ_Options pOptions, int dmNum);
```

3. The Interface Procedures and Functions

Description of the interface procedures is below.

3.1 Connect_AZ _Decoder

PAZ_Decoder Connect_AZ_Decoder (int maxrowcount, int maxcolcount);

Description.

Function generates new instance of class encapsulating the decoder functionality.

Parameters.

Maximum of horizontal and vertical image sizes.

Return value.
Pointer to decoder in success, or NULL otherwise.

3.2 Disconnect_AZ_Decoder

void Disconnect_AZ_Decoder(PAZ_Decoder & pDecoder);

Description.
Procedure destroys decoder class and frees memory.
Parameter.
Pointer to decoder. Decoder should be connected.

3.3 Create_AZ_Options

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

PAZ_Options Create_AZ_Options (PAZ_Decoder pDecoder,TAZ_OptMode 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.

3.4 Delete_ AZ_Options

void Delete_AZ_Options (PAZ_Options & pOptions);

Description.
The function destroys a handler.

Parameters.
- Handler of decoder with options.

3.5 Decode_AZ_Bits

```
int     Decode_AZ_Bits  (       PAZ_Options     pOptions,
                        int     actualrowcount,
                        int     actualcolcount,
                        TRow*   prows);
```

Description.

The function processes an image and fills Image Info and array of Symbol Infos.

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.

If the only symbol was decoded then Rejection Reason may be not 0.

3.5.1 Get_AZ_ImageInfo

PAZ_ImageInfo Get_AZ_ImageInfo (PAZ_Options pOptions);

Description.

The function returns image info.

Return value.

Pointer to Image Info.

3.5.2 Get_AZ_Info

PAZ_Info Get_AZ_Info (PAZ_Options pOptions, int dmNum);

Description.

The function returns Aztec code symbol info.

Parameters.

- Handler of decoder with options
  - Number (index) of decoded symbol in image. Only 0.

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

Return value.

Pointer to Symbol Info.

4. Demo application - GUI

All major Library features are illustrated by the C# application GUI:

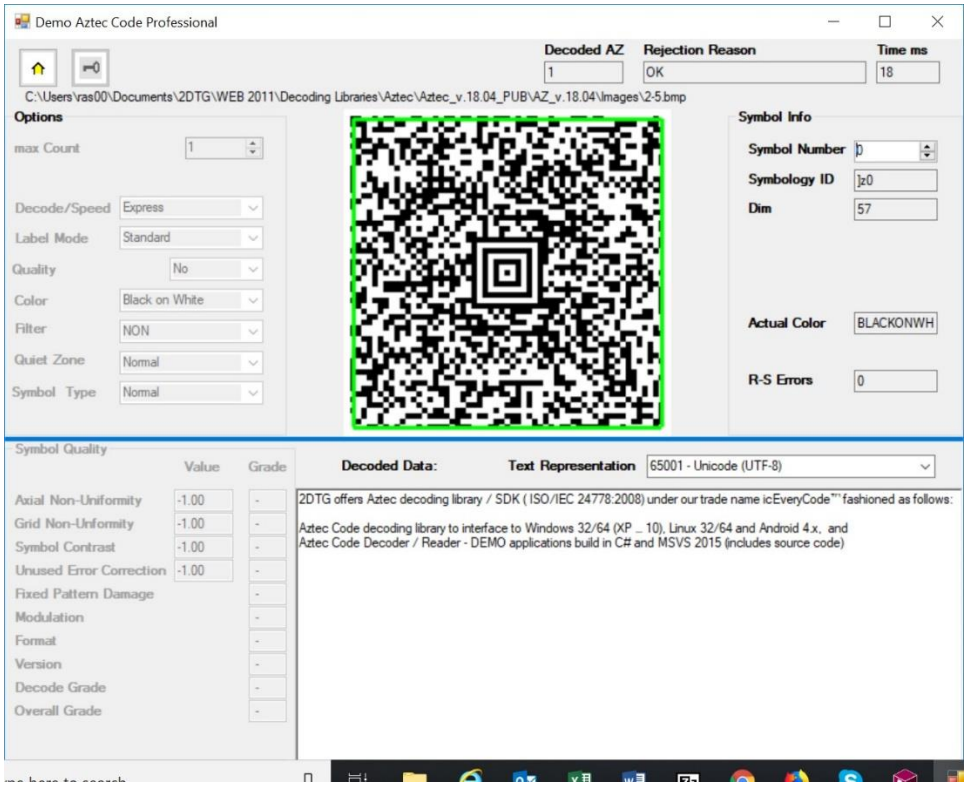

Image Info:

- **Decoded AZ** – number of Aztec symbols decoded within the image
- **Rejection Reason -** in some cases decoding library can return certain error codes and textual information, associated with the decoding process (this parameter characterizes the whole image if it contains more than 1 symbol). They are as follows:

  AZ_NotFound_Error

  Reed-Solomon_Error
- **Time** (ms) – total decode time (all Aztec symbols within the image)

Decode Window – displays all Aztec Codes found. Decode data on each code are available by "clicking" corresponding symbol within this Window.

Symbol Info:

- **Symbol Number** – Image Number to display (starting from "0") – applicable only for multiple symbols in the image.
- **Symbology ID** – indicates whether symbology is "regular" or GS1

- **Dim -** Aztec Code dimensions – in number of modules
- **Actual Color** – shows if the color of displayed symbol is regular or inversed
- **R-S Errors** – number of Reed-Solomon errors in displayed decoded symbol

Decoded Data – decoding result

5. 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
- XML-file having syntax: **[Product Name].xml**, for example: **DM Decoding 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.