Data Matrix Decoding SDK
(Professional, DPM, Enterprise editions)

User's Guide

Table of Contents

1. Introduction.

1.1 Scope

This document is applicable to the **Professional, DPM** and **Enterprise** editions of the Data Matrix Decoding SDK.

SDK is notated as **DM_XXX_YY**, where **XXX**=DPM|PRO|EP, **YY**=32|64, and notation "32|64" means 32 bit or 64 bit version.

Library interface for all three editions is uniform for Windows (XP…10), Linux, and certain embedded platforms. Both static and dynamic libraries are available.

The library is designed to decode Data Matrices ECC200 in accordance with ISO/IEC 16022 Symbology specification. Symbol quality assessment is provided in accordance with ISO/IEC 15415 and ISO/IEC TR29158.

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

1.2 Normative references

ISO/IEC 16022 - Symbology specification - Data Matrix
ISO/IEC 15415 - Symbol quality - Bar code print quality test specification — Two-dimensional symbols
ISO/IEC TR29158 - Direct Part Mark Quality Guideline
AIM DPM Quality Guideline

1.3 SDK composition

Decoding SDK contains:

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

1.4 Features Description

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

Data Matrix SDK				
Features	**EDITION**			**Description**
	Professional	**DPM**	**Enterprise**	
Print Quality Metrics (Quality Parameters)	√		√	Quality Parameters assessment in accordance with ISO 15415
Dot Peen Data Matrix decoding (DPM)		√	√	provides DPM (including Dot Peen) decoding in accordance with AIM DPM Quality Guideline
Preprocessing Filters		√	√	provides for two types of filters: • **Sharpening Filters**, recommended for low contrast and blurred images, including Adaptive (Auto) Filter and Musk Filter, and Sharp1, Sharp2 iterative filters; • **BWR Filter**, compensating for size irregularities in DataMatrix cells
Decode / Speed Selector	√	√	√	Provides for few speed/robustness options: • **Ultimate+** - designed to improve decoding for highly uneven pattern illumination samples and other particularly challenging images; • **Ultimate** – close to Ultimate+ in robustness, but slightly faster; • **Regular** – compromise between robustness and speed;
Multiple DataMatrix decoding	√		√	decodes up to 400 barcodes within one image via variable settings
Quiet Zone	√	√	√	allows for reduced Quiet Zone of Data Matrix
Inverse Color DataMatrix decoding	√	√	√	allows to speed up symbol decoding when its color can be defined in advance
Mirror DataMatrix decoding	√	√	√	provides for decoding of a "mirrored" Data Matrix symbol

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 TDM_OptMode - the set of decoder options,
struct TDM_ImageInfo - features of decoded image,
struct TDM_Info - features of decoded symbols,
struct TDM_Quality - Quality Parameters of decoded symbols.

2.1 Decoder options

```
/// decoder option modes
struct TDM_OptMode
{
  int maxDMCount;   //!< from 1 to 100. 1 by default
  int cellColor;    //!< CL_ANY by default
  int mirrorMode;   //!< MM_NORMAL by default
  int speedMode;    //!< SP_ROBUST by default
  int qualityMask;  //!< DM_QM_NO by default
  int labelMode;    //!< LM_NORMAL by default
  int timeOut;      //!< timeOut in mls. Timeout <= 0  means infinite timeout
  int filterMode;   //!< FM_NON by default
  int qzMode;
};
```

2.2 Image info

```
/// results of decoding the whole Image
struct TDM_ImageInfo
```

```
{
  int DMCount;        //!< number of well decoded symbols within image
  int RejectionReason;//!< not DM_RR_OK if no one matrix has been well
decoded
  int BreakReason;    //!< 0 - normal termination, 1 - termination by time-
out
 };
```

ImageInfo.DMCount = 1 if any Rectangle-shaped object was detected in image.
It happens if
RejectionReason = DM_RR_OK,
RejectionReason = DM_RR_BYCRIT,
RejectionReason = DM_RR_REEDSOLOMON.
If DMCount = 1 the rectangle Corners and some of Quality Parameters are defined.

BreakReason let us know whether the time out or user break happened (for embedded platforms only).

2.3 Symbol info

Each decoded symbol is described by the following structures:

```
/// Data Matrix Quality Parameters
struct TDM_Quality
{
    float symbol_contrast;
    float axial_nonuniformity;
    float grid_nonuniformity;
    float fixed_pattern_damage;     //!< the aggregate grade
    float unused_error_correction;
    float vertical_print_growth;
    float horizontal_print_growth;

    float symbol_contrast_grade;
    float axial_nonuniformity_grade;
    float grid_nonuniformity_grade;
    float fixed_pattern_damage_grade;
    float unused_error_correction_grade;
    float modulation_grade;
    float decode_grade;             //!< 4 if DM was successfully decoded
    float overall_grade;            //!< minimum of grades
};

/// result of decoding of each Data Matrix symbol in image

struct TDM_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            VDim, HDim;  //!< vertical and horizontal dimensions of Data
Matrix
  int            saTotalSymbolsNumber //!< structured append: total number of
matrices
  //!< value 0xff indicates ReaderProgramming - a special case
                ,saSymbolPosition     //!< current matrix index
                ,saFileID1            //!< file identifier 1
                ,saFileID2;    //!< file identifier 2
  int            mirrored;     //!< true if mirrored Data Matrix
  int            dotpeenstage; //!< true if dot peened Data Matrix
  int            matrixcolor;  //!< detected color of Data Matrix
  TDM_Quality    quality;      //!< symbol Quality Parameters
};
```

2.4 The Constants

```
enum CELL_COLOR{
 CL_BLACKONWHITE = 1,
 CL_WHITEONBLACK = 2,
 CL_ANY          = 3
};

enum MIRROR_MODE{
 MM_NORMAL   = 1,
 MM_MIRROR   = 2,
 MM_ANY      = 3
};

enum Decoder_SPEED{
 SP_ROBUST   = 0,
 SP_FAST     = 1,
 SP_GRID_ADJUSTMENT = 2,
 SP_EQUALIZATION    = 3, //!< re-equalization of regions probable Data Matrix
 SP_EQUAL_GRADJ     = 4,
 SP_ACCURATE        = 5,
 SP_ACCURATEPLUS    = 6
};

/// the aliases:
enum DM_SPEED{
 DMSP_ULTIMATEPLUS  = SP_ACCURATEPLUS,//!< most careful and time-expensive
 DMSP_ULTIMATE      = SP_ACCURATE,    //!< more careful and time-expensive
 DMSP_REGULAR       = SP_EQUAL_GRADJ, //!< recommended ratio "speed/quality"
 DMSP_EXPRESS       = SP_ROBUST       //!< basic algorithm (more fast)
};

enum LABEL_MODE{
 LM_STANDARD = 0,      //!<-ISO 16022
 LM_DOTPEEN  = 1,
 LM_FAX      = 2,
```

```
 LM_ST_DOT    = 3       //!< Combines Standard & Dotpeen
};

/// \enum QUALITY_MASK bits of mask:
enum QUALITY_MASK{
 DM_QM_NO        = 0X0000,
 DM_QM_AXNU      = 0X0001,
 DM_QM_PRGR      = 0X0002,
 DM_QM_SYMCTR    = 0X0004,
 DM_QM_CELLINFO  = 0X0008,
 DM_QM_ALL       = 0x7FFF
};

enum FILTER_MODE{
 FM_NON          = 0, //!< No filter
 FM_SHARP1       = 1, //!< First  Filter Mode (recursive sharpening)
 FM_SHARP2       = 2, //!< Second Filter Mode (recursive sharpening)
 FM_SHARPMASK    = 3, //!< Sharpening Mask Filter
 FM_AUTO         = 4  //!< Auto selection of sharpening parameters
,FM_BWR          = 5  //!< Bar Width Reduction (spaces enlargement)
,FM_SM_BWR       = 6  //!< Sharpening Mask + Bar Width Reduction
};

enum QRQZ_MODE{
 DMQZ_NORMAL    = 0 //!< allows QZ>= 5.7 pixels
,DMQZ_SMALL     = 1 //!< allows QZ>= 4.5 pixels, affects speed and robustness
};

enum DM_REJECTION_REASON{
 DM_RR_OK             = 0,
 DM_RR_NON            = 1,
 DM_RR_NODATAMATRIX   = 2,
 DM_RR_BYCRIT         = 3,
 DM_RR_REEDSOLOMON    = 5,
 DM_RR_NOMEMORY       = 99,
 DM_RR_UNKNOWN        = 100,
 DM_RR_DISCONNECTED   = 200
};

enum DM_BREAK_REASON{     //!< invalid anyware except of TI platform
//----------------------
 DM_ALL_INSPECTED     = 0 //!< no breaks occurred
,DM_TIMEOUT           = 1 //!< termination by time out
,DM_TERMINATED        = 2 //!< termination by user break
};
```

2.5 Type definitions

```
typedef void*           PDM_Decoder;    //!< handler of Data Matrix Decoder
typedef void*           PDM_Options;    //!< handler of Decoder Options
typedef TDM_ImageInfo*  PDM_ImageInfo;  //!< pointer to Image Info
typedef TDM_Quality*    PDM_Quality;    //!< pointer to symbol Quality
typedef TDM_Info*       PDM_Info;       //!< pointer to symbol Info
```

```
typedef unsigned char*  TRow;          //!< pointer to bitmap line

/// The function creates Data Matrix Decoder and returns Decoder handler
typedef PDM_Decoder (stdcall *TConnect_DM_Decoder)(int maxrow, int maxcol);

/// The function destroys Data Matrix Decoder
typedef void        (stdcall *TDisconnect_DM_Decoder)(PDM_Decoder &pDecoder);

/// The function creates Decoder Options and returns Options handler
typedef PDM_Options (stdcall *TCreate_DM_Options)(PDM_Decoder pDecoder,
TDM_OptMode optmode);

/// The function destroys Decoder Options
typedef void        (stdcall *TDelete_DM_Options)(PDM_Options &pOptions);

/// The function decodes array ppbits with given Options
typedef int (stdcall *TdecodeDM_Bits)(PDM_Options pOptions, int rowcount, int
colcount, TRow* ppbits);

/// The function returnes the ImageInfo of last decoded Image
typedef PDM_ImageInfo (stdcall *TGetDM_ImageInfo)(PDM_Options pOptions);

/// The function returnes the DM_Info(dmNum)
typedef PDM_Info      (stdcall *TGetDM_Info)(PDM_Options pOptions, int
dmNum);
```

3. The Interface Procedures and Functions

Description of the interface procedures is below.

3.1 Connect_DM_Decoder

PDM_Decoder Connect_DM_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_DM_Decoder

void Disconnect_DM_Decoder(PDM_Decoder & pDecoder);

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

3.3 Create_DM_Options

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

PDM_Options Create_DM_Options (PDM_Decoder pDecoder,TDM_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_ DM_Options

void Delete_DM_Options (PDM_Options & pOptions);

Description.
The function destroys a handler.

Parameters.
- Handler of decoder with options.

3.5 DecodeDM_Bits

```
int     DecodeDM_Bits   (       PDM_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 DM_RR_OK.

3.5.1 GetDM_ImageInfo

PDM_ImageInfo GetDM_ImageInfo (PDM_Options pOptions);

Description.

The function returns image info.

Return value.

Pointer to Image Info.

3.5.2 GetDM_Info

PDM_Info GetDM_Info (PDM_Options pOptions, int dmNum);

Description.

The function returns Data Matrix symbol info.

Parameters.

- Handler of decoder with options
- 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 Symbol Info.

4. GS1 Compliance

GS1 DataMatrix uses a special start combination to differentiate the GS1 DataMatrix symbol from the other Data Matrix ECC 200 symbols. This is achieved by using the Function 1 Symbol Character (FNC1) in the first position of the data encoded. It enables scanners to process the information according to the GS1 System Rules.

The FNC1 (ASCII 232) is encoded in two separate ways within GS1 DataMatrix:

- Start character
- Field Separator (to seperate varible length article identifiers)

In accordance with ISO/IEC 15424 - Data Carrier Identifiers (including Symbology Identifiers), the Symbology Identifier (the first three characters transmitted by the scanner indicating symbology type) **]d2** specifies that the symbol read is a GS1 DataMatrix symbol while **]d1,** for example, specifies regular ECC 200 symbol.

2DTG's decoding library returns Symbology Identifier that can be used by GS1 users when building their applications.

In our example of Library usage in Windows OS (DEMO Application) – Section 3.6 of this User's Guide - Symbol Info is represented in variable "**PDM_Info pdminfo**".

Decoding GS1 Data Matrix (on the right) returns the result, as follows:
pdminfo->pch = "010345312000001117120508l0ABCD1234\x1D4109501101020917";

The Symbology Identifier is stored in preamble of pch with negative indexes [-3..-0].

You can extract a value of Symbology Identifier by following operators:

```
char Symbology_Identifier[4];
strncpy(Symbology_Identifier,(char*)&(pdm_info->pch[-3]),3);
Symbology_Identifier[3] = 0;
```

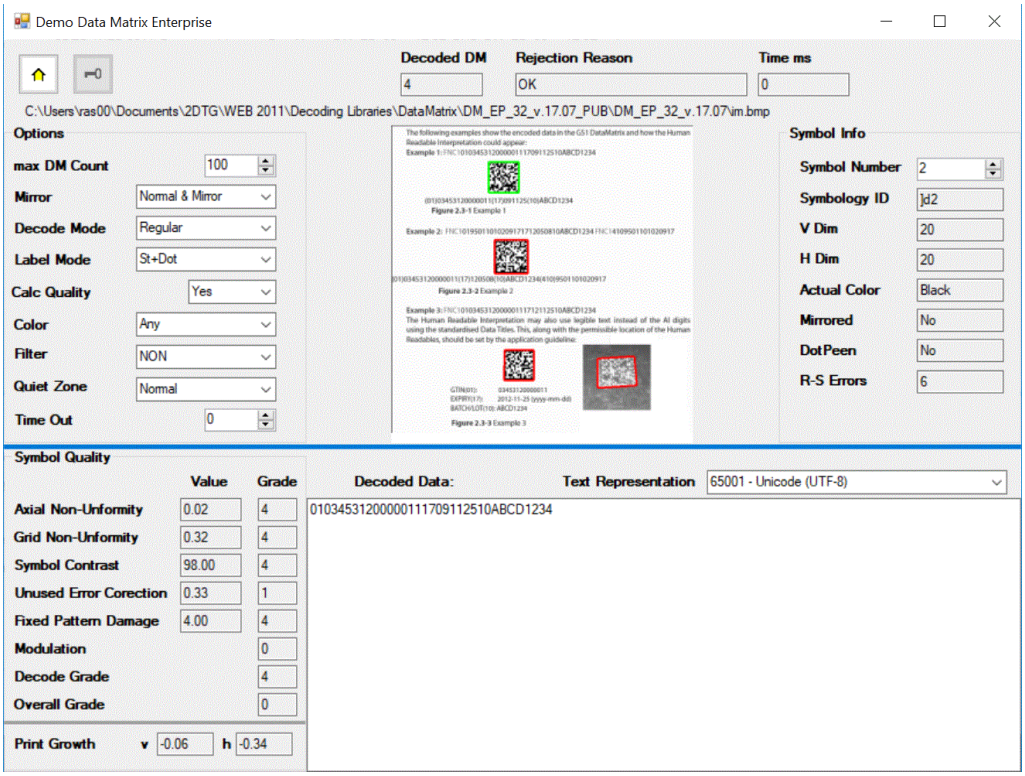
In other words in case of GS1 Data Matrix in decoded pch (from index -3) we receive:

-3..0..
"]d2010345312000001117120508l0ABCD1234\x1D4109501101020917"

while the input string was (Second FNC1 here is used like fields separator):

FNC1010345312000001117120508l0ABCD1234**FNC1**4109501101020917

GUI picture below illustrates Symbology Identifier feature of the Library: 3 symbols represent GS1 Data Matrix (Symbology Identifier =]d2) and one – regular ECC 200 (Symbology Identifier =]d1):



5. Print Quality Metrics (Quality Parameters)

2DTG offers Print Quality Metrics (PQM) and Quality Parameters (QM) evaluation module as part of the decoding library. It is based on the ISO/IEC 15415 Standard and ISO/IEC TR 29158 Technical Report, and it can be used both in barcode verifiers and barcode readers.

There are some important considerations, however, which must be kept in mind when using this module:

1. For QUALITY PARAMETERS evaluation it can be utilized only when used in Barcode Verifier subject to compliance with the articles 7.2-7.3 of the 15415 Standard and article 6 of the TR 29158.
   Barcode verifiers ensure codes are marked correctly and meet an industry's—rather than an individual producer's—quality threshold. ISO/IEC 15415 demands that image capturing and decoding should meet the whole set of strict requirements to satisfy this quality threshold.

2. In all other cases - particularly when used in a regular barcode reader – it can serve only as an evaluation tool for PRINT QUALITY METRICS. Accordingly, it is not recommended to use PQM data for decision making on accepting/rejecting evaluated symbols based on minimum acceptable grade when image captured by the barcode reader – not Verifier.

   At the same time, this tool may be very helpful providing that all readings are performed in similar conditions: lighting environment, reading distance, reading angle, aperture, substrate characteristics. In this case PQM can be used for process control and improvement as well as "reasonable" anticipation whether a generic reader will be able to successfully read these codes along the supply chain (with the understanding, of course, that PQM cannot gauge with 100% certainty how two different barcode readers will handle the same code).

3. Aperture is one of the most important parameters for PQM grading, particularly for Modulation and Grid Non-Uniformity. Unlike the optical aperture, this term refers to the certain size circle comprised of pixels, within which light is reflected to the verifier's sensor. Aperture is so important that (according to the above-mentioned Standards) a symbol grade is only meaningful if it is reported in conjunction with the illumination and aperture used. It should be shown in the format ***grade/aperture/light/angle (***ISO/IEC 15415)**.**

   Aperture size is specified by the user application specification to suit the *X* dimension *(module size)* of the symbol and the intended scanning environment. Unfortunately, both scanning environment and aperture size is normally not known in advance when performing PQM evaluation using barcode reader.

   According to ISO/IEC 15415, "Matrix symbol grading shall be carried out using a synthesized aperture of 0,8*X* diameter. In an application where symbols of differing *X* dimensions will be encountered, all measurements should be made with the aperture appropriate to the smallest *X* dimension to be encountered". For example, GS1 organization recommends the *X* dimension to be between 10 and 20 mils for the trade items scanned in general retain POS. It means that aperture size for such "symbols family" should be 8 mils regardless of actual *X* dimension, if evaluated within the same process control.

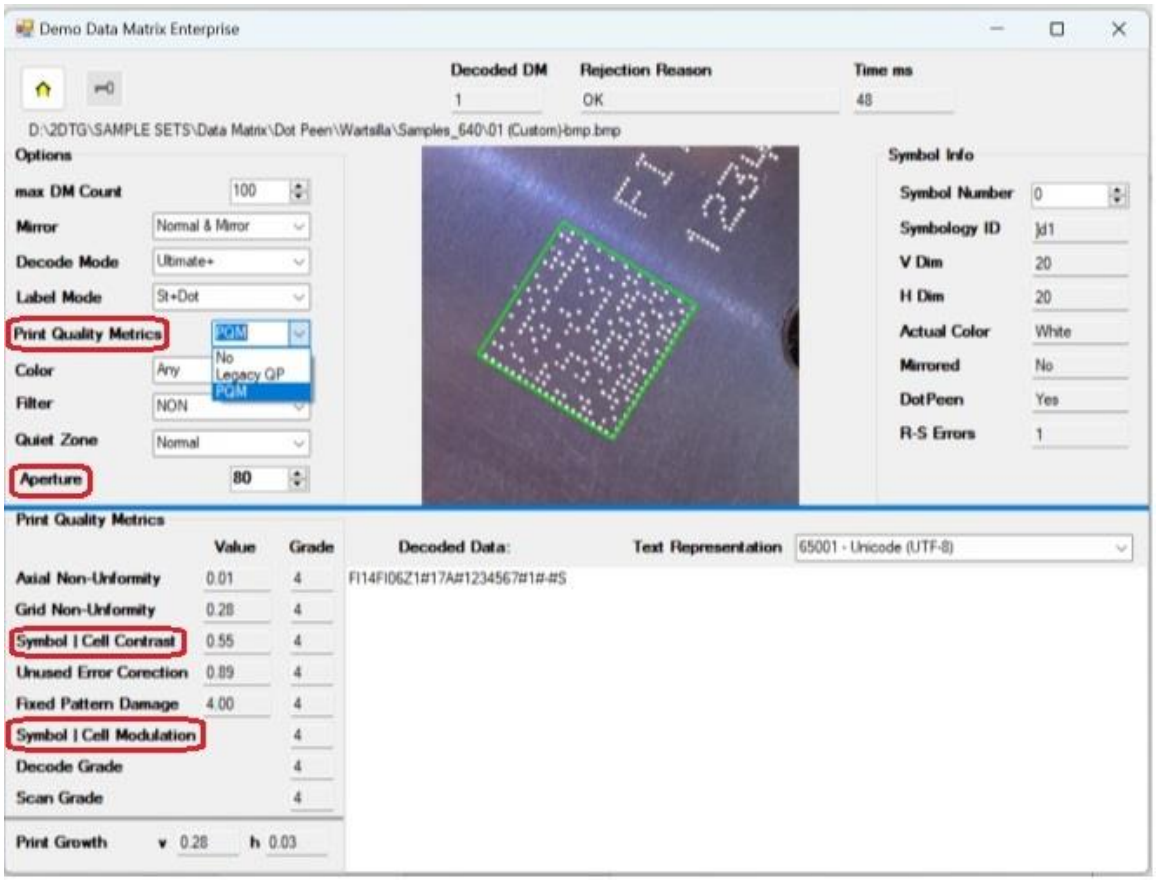
   Our library provides an option of setting the Aperture for evaluating symbols of differing *X* dimensions encountered into the same application specification (like GS1 symbols, for instance). The default setting is 80% of module size.

4. PQM/QP evaluation for DPM/Dot Peen symbols is based on ISO/IEC TR 29158. There are 3 main differences in parameters grading (and evaluation algorithm, of course) for Dot Peen samples:
   - Symbol Contrast is replaced with Cell Contrast (CC)

- Modulation is replaced with Cell Modulation (CM), and
- Fixed Pattern Damage evaluation is modified to reflect on changing the calculation of the average grade of the segments (re-naming average grade as "distributed damage grade" at the same time).

The remainder of the grading calculations are from 15415.

These changes are reflected in the user interface and the GUI of our Demo application:

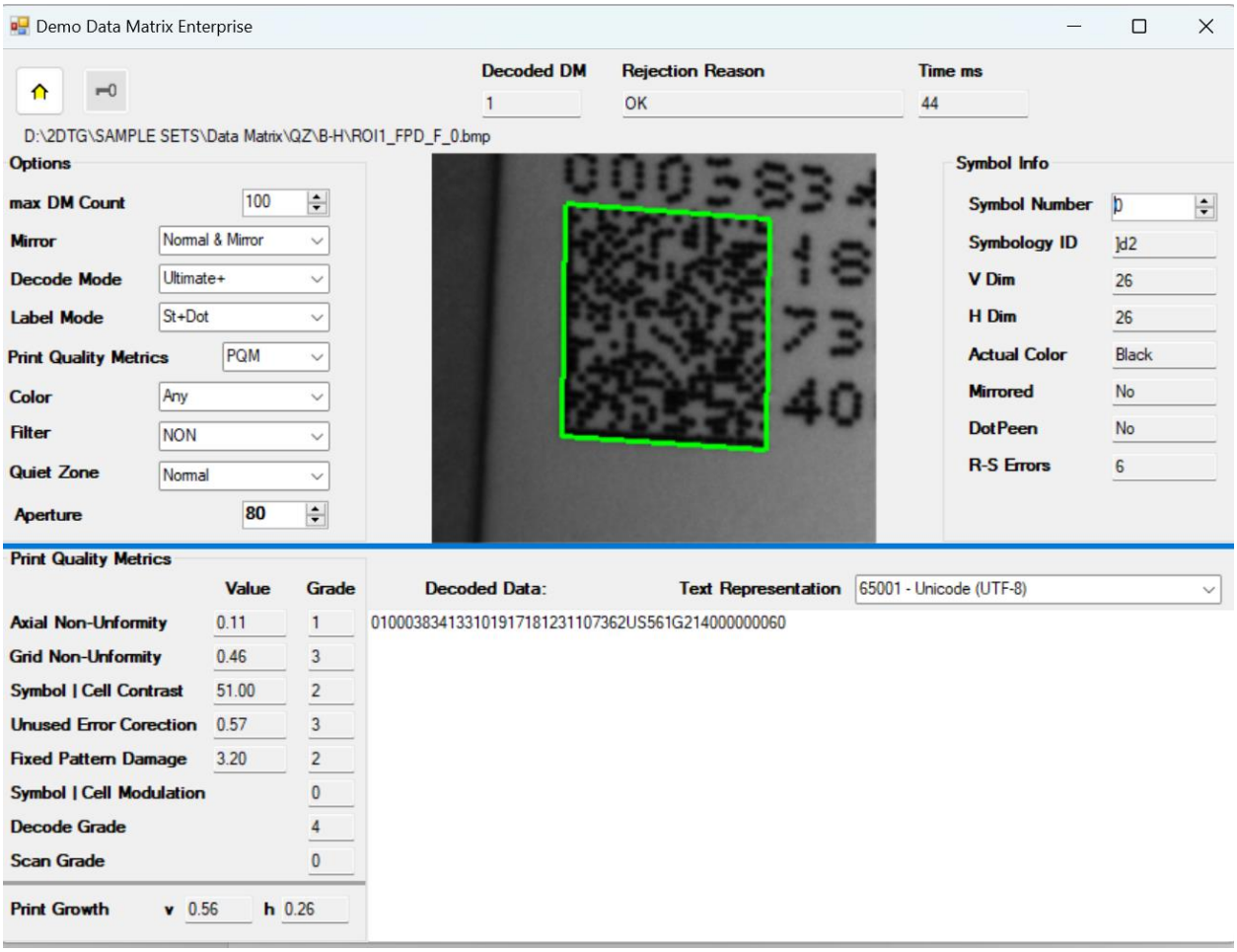


6. Demo applications

Decoding Library comes with the Demo applications written using C# and C++ languages in MSVS development environment.

6.1 C# Demo application – GUI

GUI illustrates all major features of the Library as well as the use of different options for decoding:



Decode Settings Options (described in the Section 1.4):

- **Max DM count** – number of Data Matrix symbols within an image (if known in advance) – default number = 100, total – 400;
- **Mirror** – Normal, Mirror, Normal&Mirror (default, if not known in advance)
- **Decode/Speed –** Ultimate+ (Default), Ultimate, Regular
- **Label Mode** – Standard, Dot Peen (DPM), St+Dot (default)
- **Print Quality Metrics (Quality Parameters):**
  - **NO**
  - **Legacy QP**
  - **PQM**
- **Color** – Black, White, Any (default, if not known in advance)
- **Filter** – default – "AUTO" (see Section 5.4 for detail)
- **Quiet Zone** – Normal (per ISO 16022), Small (default – "Normal")
- **Aperture (80%)** of *X* - default

Overall decode info:

- **Decoded DM** – number of Data Matrix decoded in this image
- **Rejection Reason** – returns decode result:
  - **"OK"** successful decoding (DM_RR_OK = 0) **or**

  Error Code - in some cases decoding library can return certain error codes associated with the decoding process. They are as follows:
  - **Error Code 1** – (DM_RR_NON = 1) – no "structured formations" found within the image
  - **Error Code 2** – (DM_RR_NODATAMATRIX= 2) - no "matrix-like formations" found within the image
  - **Error Code 3** – (DM_RR_BYCRIT = 3) - alternating pattern is incorrect (dark and light modules in the finder pattern do not meet alternation criteria)
  - **Error Code 5** – (DM_RR_REEDSOLOMON = 5) – excessive number of Reed-Solomon error
- **Time** (ms) – total decode time

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 Data Matrix identifier for displayed symbol
- **V Dm, H Dm** – Data Matrix dimensions (Vertical, Horizontal) – in number of modules
- **Actual Color** – shows if the color of displayed symbol is regular or inversed
- **Mirrored** – shows if displayed symbol is mirrored or not
- **DotPeen** – shows if displayed symbol was decode using Dot Peen algorithm or Standard one
- **R-S Errors** – number of Reed-Solomon errors in displayed decoded symbol

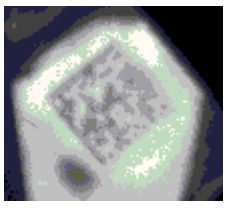
Print Quality Metrics – results of the symbol quality assessment in accordance with ISO/IEC 15415
Print Growth - calculated per ISO/IEC 15415

6.2 C++ Demo application - Example of Library usage

```
// example of Windows application

// ======================== variables ============================

#include "DMPro_Types.h"
```

```
   int            rowcount, colcount; // The Img dimentions
   TRow           pbits[4000];        // array of pointers to bitmap lines
                                      // (input parameter for decoding)
   void*          pdecoder;
   PDM_Options    poptions;
   TDM_OptMode    optmode;
   PDM_ImageInfo  pimageinfo;
   PDM_Info       pdminfo;

   TConnect_DM_Decoder      Connect_DM_Decoder;
   TDisconnect_DM_Decoder   Disconnect_DM_Decoder;
   TCreate_DM_Options       Create_DM_Options;
   TDelete_DM_Options       Delete_DM_Options;
   TdecodeDM_Bits           DecodeDM_Bits;
   TGetDM_ImageInfo         GetDM_ImageInfo;
   TGetDM_Info              GetDM_Info;

   HINSTANCE      dllinstance;
   int            res, i, DecodedMatrixNo;

…

// ============================ program ================================

  dllinstance = LoadLibrary("..\\Lib\\DM_PRO_32.dll");
//dllinstance = LoadLibrary("..\\Lib\\DM_PRO_64.dll"); //in 64-bit
applications

  if (dllinstance!=NULL) {
   Connect_DM_Decoder    = (TConnect_DM_Decoder
)GetProcAddress(dllinstance,"Connect_DM_Decoder");
   Disconnect_DM_Decoder = (TDisconnect_DM_Decoder
)GetProcAddress(dllinstance,"Disconnect_DM_Decoder");
   Create_DM_Options     = (TCreate_DM_Options
)GetProcAddress(dllinstance,"Create_DM_Options");
   Delete_DM_Options     = (TDelete_DM_Options
)GetProcAddress(dllinstance,"Delete_DM_Options");
   DecodeDM_Bits         = (TdecodeDM_Bits
)GetProcAddress(dllinstance,"DecodeDM_Bits");
   GetDM_ImageInfo       = (TGetDM_ImageInfo
)GetProcAddress(dllinstance,"GetDM_ImageInfo");
   GetDM_Info            = (TGetDM_Info
)GetProcAddress(dllinstance,"GetDM_Info");
  }

  if (Connect_DM_Decoder != NULL) {

    // ==== construct decoder:
    pdecoder = Connect_DM_Decoder(4000,4000);

    // ==== Assign option modes
    optmode.maxDMCount  = 1; // single, 100 - maximum
    optmode.speedMode   = DMSP_REGULAR;
    optmode.cellColor   = 3; // 1 - BlackOnWhite, 2 - WhiteOnBlack, 3 - any
```

```
    optmode.mirrorMode  = 1; // 1 - Normal, 2 - Mirror, 3 - both
    optmode.qualityMask = 0; // 0xFFFF – all Quality Parameters
    optmode.labelMode   = 0; // 0-standard, 1-dotpeen, 2-fax, 3-
                                Standard+Dotpeen
    optmode.timeOut     = 0; // 0 ms
    optmode.filterMode  = 0; // don't filter (1,2, 3 - sharpening)

    // ==== Construct the options:
    poptions = Create_DM_Options(pdecoder, optmode);

    while(...) { // ========== begin decode loop:

       //  ... Load new image into pbits

       res = DecodeDM_Bits(poptions,rowcount,colcount,pbits); //decode the
array

       pimageinfo = GetDM_ImageInfo(poptions);
       // display pimageInfo]

       DecodedMatrixNo = pimageinfo->DMCount;

       if ((DecodedMatrixNo > 0){
         for (i=0; i<DecodedMatrixNo; i++){
           pdminfo = GetDM_Info(poptions,i);
           // display pdmInfo [i]
         }
       }
     } // ============ end of decode loop

     Delete_DM_Options(poptions);

     Disconnect_DM_Decoder (pdecoder);

     FreeLibrary(dllinstance);
```



7. Applying Pre-processing Filters

Data Matrix decoding library, Enterprise edition comes with of optional pre-processing filters:

- **Sharpening filters** - **Adaptive (Auto) Filter** and Musk Filter (**SharpMask**) recommended for low contrast and blurred images (Sample of the image that may require sharpening is shown here (decodable only after applying **SharpMask Filter**)), and

- "**Print Correction Filter"** or "**BWR filter**" - designed to compensate for the printing conditions ("overprinting") of some Data Matrix barcodes, having substantial irregularities in the printed module size and/or Grid Non-Uniformity (GNU).

  ISO standard specifies required dimensions and tolerances in the final printed Data Matrix symbol. In real life, however, after the code is printed the dark cells may end up greater than the light ones due to a number of factors, but, most probably, due to the excessive ink spread in dark regions. If this "spreading" is too big (beyond the ISO standard), datamatrix decoding software may not be capable of "reading" the bar code (this image at right illustrates also the additional "printing" problem – the irregularities in the alternating pattern or even its "warping").

  Similarly, the wear of the printing machine may result in displacement of the actual grid nodes towards their nominal positions in each cell of Data Matrix, causing it to become "unreadable".

  Using "**BWR Filter"** allows to decode such codes, which are, otherwise, "not readable".

All filters are supposed to be applied to the captured image before decoding procedure if the corresponding option is chosen in the initial settings.

Important:

1. Caution shall be taken when applying the filters. If it is applied to the "regular" (reasonable quality) image it can make it undecodable. Only **Adaptive (Auto) filter** can be safely applied to any image – it does not degrade the symbol. That is why it is recommended always try the regular decoder first and apply filter only if it fails.

2. If Print Quality Metrics is to be evaluated – NO filter should be set in the decoder settings.

8. 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.