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.

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 standard.

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
AIM DPM Quality Guideline

1.3 Features Description

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

Data Matrix SDK				
Features	**EDITION**			**Description**
	Professional	**DPM**	**Enterprise**	
Data Matrix Quality Parameters	√	√	√	Quality Parameters assesment 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	√		√	Provides for three speed/robustness options: • **Regular Mode** – for most (including DPM)

Selector				images - combines high success decode rate with high speed; • **Ultimate** – for the particularly challenging images (increases success decode rate by ~ 7%, but decode time may also increase by); • **Express Mode** - higher decoding speed (~15% faster than Regular mode, but success decode rate might be ~15% lower) - for the applications where decoding time is critical and image quality is reasonably good
Multiple DataMatrix decoding	√		√	decodes up to 400 barcodes within one image via variable settings
Allowable image size (pixels)	1200 x 1600	640 x 844	5000 x 8192	
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
FAX transmitted DataMatrix decoding		√	√	decodes Data Matrix symbols located within a FAX-transmitted or Tiff images

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 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-equalizing the regions of probable Data Matrix
 SP_EQUAL_GRADJ     = 4
,SP_ACCURATE        = 5
};

/// the aliases:
enum DM_SPEED{
 DMSP_ULTIMATE      = SP_ACCURATE,    //!< most accurate but time-consuming
 DMSP_REGULAR       = SP_EQUAL_GRADJ, //!< recommended ratio "speed/quality"
 DMSP_EXPRESS       = SP_ROBUST       //!< basic algorithm (faster than
regular)
};

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 =
"]d2010345312000001117120508 10ABCD1234\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..
"]d2010345312000001117120508 10ABCD1234\x1D4109501101020917"

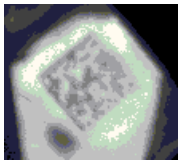
while the input string was (Second FNC1 here is used like fields separator):

FNC1010345312000001117120508 10ABCD1234**FNC1**4109501101020917

5. Applying Pre-processing Filter

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

- **Sharpening filters** - Adaptive (Auto) Filter and Musk Filters (Sharp1, Sharp 2 and 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



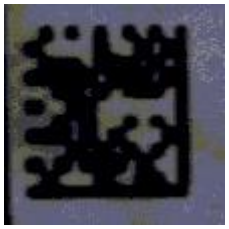
- "**Print Correction Filter"** or "**BWR filter**" - designed to compensate for the printing conditions ("overprinting") of some Data Matrix barcodes, having substantial irregularities in the printed module size and/or Grid Non-Uniformity (GNU).

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".

- **Combined Filter – "SM + BWR"** – designed to compensate both for the overprinting and fuzziness of some Data Matrix barcodes. Shall be used on barcodes with large values of "Print Growths" and having module size larger or about 5x5 pixels.



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