QR Code Decoding SDK
(Professional Edition)

User's Guide

Contents

1. Introduction.

1.1 Scope

This document is applicable to the **Professional** Edition of the QR Code Decoding SDK.

SDK is notated as **QR_PRO_YY**, where **YY**=32|64, and notation "32|64" means 32 bit or 64 bit version.

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

The library is designed to decode QR Codes (including Micro QR Code) in accordance with ISO/IEC 18004 Symbology specification and written using C++ programmic language. Symbol quality assessment is provided in accordance with ISO/IEC 15415 standard.

The Quality Parameters option is user-selectable.

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

1.2 Normative references

ISO/IEC 18004 - Symbology specification - QR Code
ISO/IEC 15415 - Symbol quality - Two-dimensional symbols

1.3 SDK composition

Decoding SDK contains:

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

1.4 Features Description

Professional Edition is an advanced product designed for a wide spectrum of applications. Its major features/advantages are as follows:

Features	Description
Micro QR Code	Allows to decode Micro QR Codes
Quality Parameters	Quality Parameters calculation in accordance with ISO 15415
Multi Mode	Decodes up to 10 barcodes within one image via variable settings
Inversed Color	Allows to choose between the regular (black-on-white) and reverse (white-on-black) images
Reduced Quiet Zone	Allows to decode symbols with reduces Quiet Zone size (down to 1 module size)
Low Contrast Filter	Pre-processing filter for low-contrast image
Mirror QR Code decoding	provides for decoding of a "mirrored" QR Code 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:

void* PQR_Decoder - handler of QR Code Decoder
void* PQR_Options - handler of Decoder Options
struct TQR_OptMode - the set of decoder options,
struct TQR_ImageInfo - features of decoded image,

struct TQR_Info - features of decoded symbols,
struct TQR_Quality - Quality Parameters of decoded symbols.

2.1 Decoder options.

```
/// decoder option modes
struct TQR_OptMode
{
int maxQRCount;   //!< belongs to range [1..10], is equal to 1 by default
int cellColor;    //!< == QRCL_BLACKONWHITE by default
int mirrorMode;   //!< QRMM_NORMAL by default
int speedMode;    //!< QRSP_ROBUST by default (not implemented)
int qualityMask;  //!< 0 – no QP, 1 – estimate QP, 0 by default
int labelMode;    //!< QRLM_NORMAL by default (not implemented)
int timeOut;      //!<timeOut in mls. Timeout <= 0  means infinite timeout
int filterMode;   //!< QRFM_NON by default
int symbology;    //!< QRC_NORMAL | QRC_MICRO | QRC_ANY
};
```

Most of these fields are listed for compatibility with Data Matrix decoder.
If we wish to find regular QR Codes only (with 3 pattern finders) we should assign field "symbology" like QRC_NORMAL .
In case of Micro QR code we set symbology = QRC_MICRO.
And finally, if we wish decode both normal and micro QR Codes in the image we set symbology = QRC_ANY

2.2 Image info.

```
/// results of decoding the whole Image
struct TQR_ImageInfo
{
  int QRCount;        //!< number of well decoded symbols within image
  int RejectionReason;//!< not QR_SUCCESSFUL if no one QR Code was decoded
int BreakReason;    //!< 0 - normal termination, 1 - termination by time-out
 };
```

ImageInfo.QRCount = 1 if any Rectangle-shaped object was detected in image.
It happens if
RejectionReason = QR_SUCCESSFUL,
RejectionReason = QR_POOR_IMAGE_QUALITY,
RejectionReason = QR_ERROR_OF_RS_CODE.
If QRCount = 1 the rectangle Corners and some of Quality Parameters are defined.

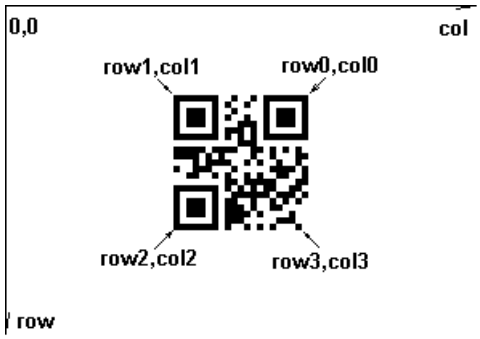
2.3 Symbol info.

Each decoded symbol is described by the following structures:

```
/// QR Code Quality Parameters
struct TQR_Quality{
  float sc;
  float an;
  float gn;
  float uec;
  float mod;   // not in use
  float fpd;     // not in use

  float scG;
  float anG;
  float gnG;
  float uecG;
  float modG;
  float fpdG;
  float frmG;
  float verG;
  float deG;
  float scanG;
};

/// result of decoding of each QR Code symbol in image
struct TQR_Info
{
  float                 rowcols[8]; //!< symbol corner coordinates
  short int             pchlen;     //!< length of decoded 2-byte array
  unsigned short*       pch2;      //!< pointer to 2-byte array (Unicode string)
  short int             RSErr;      //!< number of Reed Solomon errors
  short int             Dim;        //!< dimension of QR Code (21..177)
  int                   Version;    //!< 1..40
  char                  level;      //!< ' ', 'L', 'M', 'Q', 'H'
  int                   Mask;       //!< 1..9
  int                   Micro_QRC;   //!< ==0 for QRC,  ==1 for Micro QRC
  int      Color;       //!< 1 - black, 2 - white
  int      Mirrored; //!< 1 - yes, 0 - no
  TQR_Quality           quality;    //!< symbol Quality Parameters
};
```

Field Micro_QRC indicates the mode (0 - Normal and 1 - Micro) of current symbol.

The array rowcols is ordered as {row0, col0 . . . row3, col3}:



2.4 The Constants.

```
enum QRCELL_COLOR{
 QRCL_BLACKONWHITE = 1
,QRCL_WHITEONBLACK = 2
,QRCL_ANY         = 3
};

enum QRMIRROR_MODE{
 QRMM_NORMAL   = 1
,QRMM_MIRROR   = 2
,QRMM_ANY      = 3
};

enum QRDECODER_SPEED{
 QRSP_ROBUST   = 0
};

enum QRLABEL_MODE{
 QRLM_STANDARD = 0     //!<-ISO 18004 with restrictions
};

/// \enum QUALITY_MASK bits of mask:
enum QRQUALITY_MASK{
 QR_QM_NO       = 0X0000
,QR_QM_ALL      = 0x7FFF
};

enum QRFILTER_MODE{
 QRFM_NON         = 0  //!< no filter
,QRFM_AUTO        = 1  //!< filter with digital auto-focus
};
```

```
enum QRSYMBOL_MODE{
 QRC_NORMAL = 0     //!<- Ordinary QR Code (with 3 finder patterns)
,QRC_MICRO  = 1     //!<- Micro QR Code (1 finder pattern)
,QRC_ANY    = 2     //!<- Symbols of both type will be decoded simultaneously
};

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


enum QR_BREAK_REASON{
 QR_ALL_INSPECTED       = 0 //!< no breaks occurred
,QR_TIMEOUT             = 1 //!< termination by time out
};
```

2.5 Type definitions.

```
typedef void*               PQR_Decoder;        //!< handler of QR Code Decoder
typedef void*               PQR_Options;        //!< handler of Decoder Options
typedef TQR_ImageInfo*      PQR_ImageInfo;      //!< pointer to Image Info
typedef TQR_Quality*        PQR_Quality;        //!< pointer to symbol Quality
typedef TQR_Info*           PQR_Info;           //!< pointer to symbol Info
typedef unsigned char*      TRow;               //!< pointer to bitmap line

//
// Connect Decoder
typedef PQR_Decoder (__stdcall* TConnect_QR_Decoder)(int maxrowcount, int maxcolcount);

// DisConnect Decoder
typedef         (__stdcall* TDisconnect_QR_Decoder)(PQR_Decoder &pDecoder);

// Decode QR Bits
typedef int     (__stdcall* TDecode_QR_Bits)    (PQR_Options pOptions
                                                ,int rowCount, int colCount
                                                ,unsigned char** data);

/// The function creates Decoder Options and returns Options handler
typedef PQR_Options (__stdcall *TCreate_QR_Options)    (PQR_Decoder pDecoder
                                                       ,TQR_OptMode optmode);
```

```
/// The function destroys Decoder Options
typedef void       (__stdcall *TDelete_QR_Options)(PQR_Options &pOptions);

/// The function decodes array ppbits with given Options
typedef int (__stdcall *TDecodeQR_Bits)     (PQR_Options pOptions
                                             , int rowcount, int colcount
                                             , TRow* ppbits);

/// The function returns the ImageInfo of last decoded Image
typedef PQR_ImageInfo (__stdcall *TGetQR_ImageInfo)(PQR_Options pOptions);

/// The function returns the QR_Info(qrNum)
typedef PQR_Info      (__stdcall *TGetQR_Info)(PQR_Options pOptions, int qrNum);
```

3. The Interface Procedures and Functions

3.1 Connect_QR_Decoder

PQR_Decoder Connect_QR_Decoder (int maxrowcount, int maxcolcount);

Description.
Function generates new instance of class encapsulating the decoder functionality.

Parameters.
Maximums of horizontal and vertical image sizes.

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

3.2 Disconnect_QR_Decoder

void Disconnect_QR_Decoder(PQR_Decoder & pDecoder);

Description.
Procedure destroys decoder class and frees memory.

Parameter.
Pointer to decoder. Decoder has to be connected.

3.3 Create_QR_Options.

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

PQR_Options Create_QR_Options (PQR_Decoder pDecoder,TQR_OptMode pOptions);

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_ QR_Options.

void Delete_QR_Options (PQR_Options &pOptions);

Description.
The function destroys a handler.

Parameters.
- Handler of decoder with options.

3.5 TQR_Options::DecodeQR_Bits

```
int DecodeBitsF ( PQR_Options pOptions,
                  int actualrowcount,
                  int actualcolcount,
                  TRow* prows);
```

Description.
The function processes an image and fills Image Info and Symbol Info.

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 QR_RR_OK.

3.5.1. GeTQR_ImageInfo

PQR_ImageInfo GeTQR_ImageInfo (PQR_Options pOptions);

Description.
The function returns image info.

Parameter.
- Handler of decoder with options

Return value.
Pointer to Image Info.

3.5.2. TQR_Options::GeTQR_Info

PQR_Info GeTQR_Info (PQR_Options pOptions, int qrNum);

Description.
The function returns QR Code symbol info.

Parameters.
- Handler of decoder with optionsNumber (index) of decoded symbols in image.
If no symbols were decoded we return Info about the most probable symbol location.

Return value.
Pointer to Symbol Info.

3.6 Example of library usage in Windows.

// example of MSVS Windows application

```
#include "stdafx.h"
#include "Windows.h"
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <math.h>
#include <stdlib.h>

#include "QRPro_Types.h"
#include "LoadBMP.cpp"

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


int _cdecl _tmain(int argc, _TCHAR* argv[])
{
  PQR_Decoder    pdecoder = NULL;
  PQR_Options    pdec = NULL;
  TQR_OptMode    optmode;
  PQR_ImageInfo  pimageinfo;
  PQR_Info       pqrinfo;
  PQR_Quality    qp;

  int i;
  int res;
  int rc, cc;
char* files[] =
{
 "im1.bmp"
,"im2.bmp"
,NULL   // last member of array must be NULL
};
  char fn[256];

  unsigned char *MemBits;
  unsigned char *pbits[3000];
  char ch;
  int iAttributes = 0;
  int totalfiles = 0;
  int total=0,nOK = 0;
  int qrc, n, k;
  char GRADE[] = "FDCBA";

  clock_t totaltime, dectime;

  HINSTANCE dllinstance;

  TConnect_QR_Decoder     Connect_QR_Decoder;
  TDisconnect_QR_Decoder  Disconnect_QR_Decoder;
  TCreate_QR_Options      Create_QR_Options;
  TDelete_QR_Options      Delete_QR_Options;
  TDecodeQR_Bits          DecodeQR_Bits;
  TGetQR_ImageInfo        GetQR_ImageInfo;
  TGetQR_Info             GetQR_Info;

  totaltime = 0;
  dllinstance = LoadLibrary(L"..\\QRC_Pro.dll");
  if (dllinstance==NULL)
```

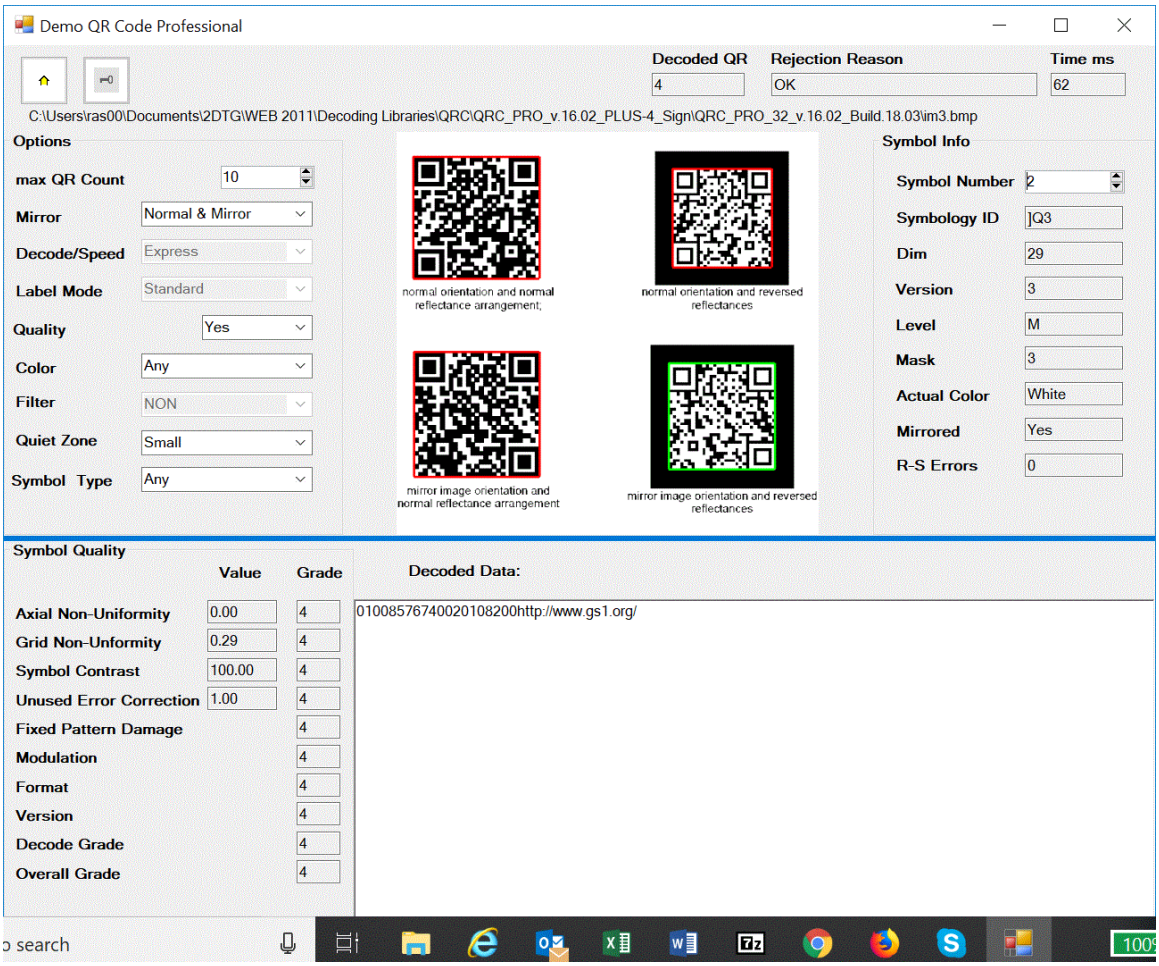
```
     goto doExit;
  Connect_QR_Decoder    = (TConnect_QR_Decoder
)GetProcAddress(dllinstance,"Connect_QR_Decoder");
  Disconnect_QR_Decoder = (TDisconnect_QR_Decoder
)GetProcAddress(dllinstance,"Disconnect_QR_Decoder");
  Create_QR_Options     = (TCreate_QR_Options
)GetProcAddress(dllinstance,"Create_QR_Options");
  Delete_QR_Options     = (TDelete_QR_Options
)GetProcAddress(dllinstance,"Delete_QR_Options");
  DecodeQR_Bits         = (TDecodeQR_Bits
)GetProcAddress(dllinstance,"DecodeQR_Bits");
  GetQR_ImageInfo       = (TGetQR_ImageInfo
)GetProcAddress(dllinstance,"GetQR_ImageInfo");
  GetQR_Info            = (TGetQR_Info
)GetProcAddress(dllinstance,"GetQR_Info");

  if (Connect_QR_Decoder==NULL)
     goto doExit;

  pdecoder = Connect_QR_Decoder(3000,3000);

  optmode.maxQRCount = 10;
  optmode.timeOut = 0; // 0 ms => don't check time-out
  optmode.qualityMask = QR_QM_ALL;
  optmode.symbology = QRC_ANY;

  pdec   = Create_QR_Options(pdecoder,optmode);
  pimageinfo = NULL;
  pqrinfo    = NULL;

  MemBits =(unsigned char*) malloc(3000*3000);
  for (i=0;i<3000;i++){
    pbits[i] = MemBits+3000*i;
  }
  for (i=0;files[i]!=NULL;i++){
     rc = cc = 3000;
     res = LoadBMP(files[i],pbits,rc,cc);
     if (res==0){
        totalfiles++;
        dectime = clock();
        DecodeQR_Bits  (pdec,rc,cc,pbits); // res = QRCount
        dectime = clock()-dectime;
        totaltime += dectime;
        pimageinfo = GetQR_ImageInfo(pdec);
        res   = pimageinfo->RejectionReason;
        qrc = pimageinfo->QRCount;
        printf("\n\n%20s ,  Time=%4d ms,  RR=%d,  QRCount=%2d"
              ,files[i], dectime, res, qrc);
        if (qrc>0)
        for (n=0;n<qrc;n++){
          pqrinfo    = GetQR_Info(pdec, n);
          qp         = &(pqrinfo->quality);
          if (pqrinfo->Micro_QRC)
            printf("\n  <M%d, %c>" ,pqrinfo->Version,pqrinfo->level);
```

```
          else
            printf("\n  < %d, %c>" ,pqrinfo->Version,pqrinfo->level);
          if (res==0){
            nOK++;
            total++;
            wprintf(L"\n  DecRes: %s",(wchar_t*)(pqrinfo->pch2));
          }
          printf("\nSC = %7.2f (%c) "   ,qp->sc,GRADE[int(qp->scG)]);
          printf(", AxNU = %7.2f (%c) " ,qp->an,GRADE[int(qp->anG)]);
          printf(", GrNU = %7.2f (%c) " ,qp->gn,GRADE[int(qp->gnG)]);
          printf(", UEC = %7.2f (%c) "  ,qp->uec,GRADE[int(qp->uecG)]);
          printf(", FPD = (%c) "  ,GRADE[int(qp->fpdG)]);
          printf(", ModG = (%c) " ,GRADE[int(qp->modG)]);
          printf(", DecG = (%c) " ,GRADE[int(qp->deG)]);
          printf(", FrmG = (%c) " ,GRADE[int(qp->frmG)]);
          printf(", VerG = (%c) " ,GRADE[int(qp->verG)]);
          printf(", ScanG = (%c) ",GRADE[int(qp->scanG)]);
        }
      }else{
        printf("\n Error LoadBMP! ");
      }
  }
  printf("\n ----------------------------------");
  printf("\n well decoded images: %d\n\n",total);

  if (pdec!=NULL)
    Delete_QR_Options (pdec);
  if (pdecoder!=NULL)
    Disconnect_QR_Decoder (pdecoder);

doExit:
  if (totalfiles==0) totalfiles=1;
  printf("\n total files %d,   QR Count= %d,    Avr.time= %d"
         ,totalfiles,nOK,totaltime/totalfiles);

  printf("\n test executed, press 'Enter' to exit>");
  scanf("%c",&ch);
  return 0;
}
```

4. GUI for C# Demo application

Decoding Library comes with the Demo applications build using C# and C++ (MSVC IDE).

GUI illustrates all major features of the Library (for multiple symbols click the symbol on the screen to see decode information):



Decode Settings Options - described in the Section 1.4.

- **Max QRC count –** number of QR (and Micro QR) symbols in the image (default – 10, maximum allowed number)
- **Mirror** – Normal, Mirror, Normal & Mirror (default, if not known in advance)
- **Quality** – quality parameters assessment ("YES" – default, "NO")
- **Color** – choose from "Black", "White", "ANY" – default
- **Quiet Zone –** Standard (4X, where X-module size); or Small (1X) - – default
- **Symbology Type –** choose from "Normal", "Micro", "Any" – default

Overall decode info:

- **Decoded QR** – number of QR Codes decoded in this image
- **Rejection Reason** – returns decode result:
  - **"OK"** successful decoding (QR_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 2** (QR_NO_IMAGE_FOUND) - no QR Code symbol has been found within the image
- **Error Code 3** (QR_POOR_IMAGE_QUALITY) - poor image quality
- **Error Code 4** (QR_ERROR_OF_RS_CODE) - excessive number of Reed-Solomon errors
- **Error Code 5** (QR_ERR_DECODE) - incorrect encodation
- **Error Code 6** (QR_ERR_MODE) - incorrect mode of encoded data
- **Error Code 10** (QR_MODE_STRUCTURED_APPEND) - structured append found

- **Time** (ms) – total decode time (all symbols within the 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** – indicates whether symbology is "regular" ( ]Q1) or GS1 ( ]Q3) - GS1 QR Code uses a special start combination to differentiate the GS1 QR Code symbol from the other QR Code symbols.
- **Dim** – QR Code dimensions – in number of modules
- **Version** – version of current Symbol (M1..M4 for Micro QR Code, 1..40 – for normal QR Code)
- **Level** – Error Protection Level (L, M, Q, H)
- **Mask** – mask index (from 0 to 7)
- **Actual Color** – shows if the color of displayed symbol is regular or inversed
- **Mirrored** – shows if displayed symbol is mirrored or not
- **R-S Errors** – number of Reed-Solomon errors in displayed decoded symbol

Symbol Quality – results of the symbol quality assessment in accordance with ISO/IEC 15415

Decoded Data – decoding result

5. GS1 Compliance

The primary use for QR Code within the GS1 System is for sharing extended packaging information.

Extended Packaging is an approach to give consumers access to additional information or services about trade items. For consumer goods, the recommendation is to use the GS1 QR Code (or GS1 DataMatrix) when brand owners want to encode a link to a website. When either barcode is scanned, the same information will be transmitted regardless of the symbology used.

The GS1 System requires the use of **symbology identifiers**. GS1 QR Code uses the symbology identifier of "]Q3" (see Figure above) for GS1 System compliant symbols that have a leading FNC1 character. The **Symbology Identifier** is placed into preamble of decoded characters, i.e. into pch2[-3], pch2[-2], pch2[-1].

GS1 QR Code *requires* the mandatory association of the **Global Trade Item Number (GTIN)** and Extended Packaging URL following **Application Identifier (AI)**. Accordingly, the **Application Identifier** (8200) – see Fig. above - indicates that the following data field contains a brand owner authorized URL to be used in mandatory association with the **GTIN AI** (01).

Additional information can be encoded as required.

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
- 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.

7. Exceptions.

The following modes are not supported:

1. Structured appends.
2. Extended Channel Interpretation (ECI)