

Instructions for LabelPrinter.dll

LabelPrinter.dll is a .NET-based SDK for barcode printer development. It is designed to handle label layout design, print preview, and output. The library is built targeting both .NET Framework 4.0 and .NET Standard 2.0.

Support:

Framework: Supports .NET Framework 2.0 - 4.6.0

Standard: Supports .NET Core 3.0 - .NET 10 (Starting from 3.1, "Core" was officially dropped in favor of direct version naming).

The calling conventions for both are consistent, with no differences in usage.

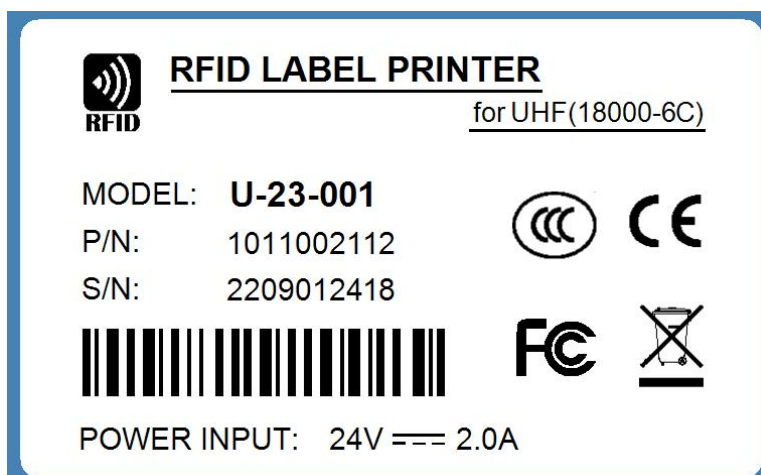
Note: The Standard framework content does not support cross-platform usage and is limited to Windows systems only!

In general, printing labels requires defining the following three parameters:

1. Printer parameters(resolution, interface, printing speed, printing blackness, etc.);
2. Label parameters (label width, height, spacing, etc.);
3. The parameters of each object that needs to be printed within the label (such as the content, location coordinates, and size of objects such as text, barcode, and images).

After defining the above parameters, various labels can be previewed or printed, supporting the generation and modification of one-dimensional codes, two-dimensional codes, text, images, graphics, RFID reading and writing objects, etc.

The following figure is an example of each object in a label:



Catalog

1. How to reference a DLL file
2. Writing Code
 - (1) Define printer
 - (2) Define labels

(3) Define each object within the label

(4) Preview Label

(5) Print labels

3. Run Code

4. Introduction to Other Functions

(1) Read LSF label template

(2) Reading TID or EPC of UHF tags

(3) Read the UID of high-frequency tags

(4) Print a blank page

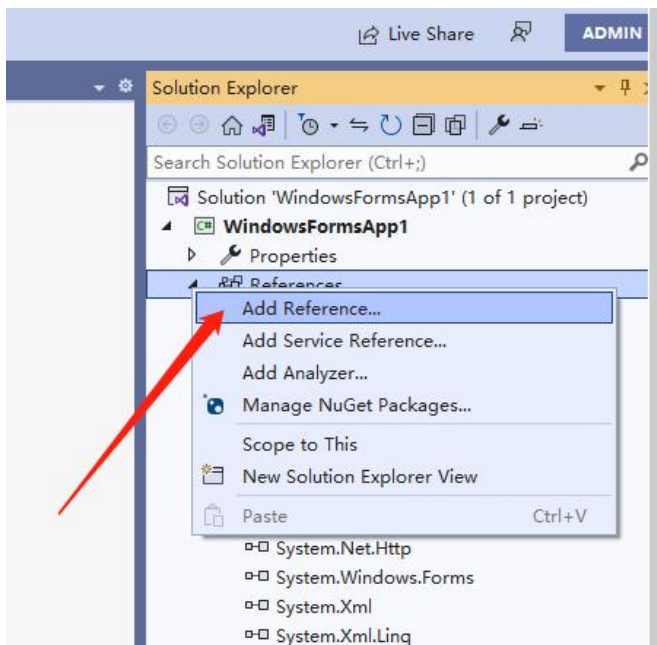
(5) Query printer status function

5. Introduction to the parameters of printer class, label class, and various object classes that need to be printed within the label

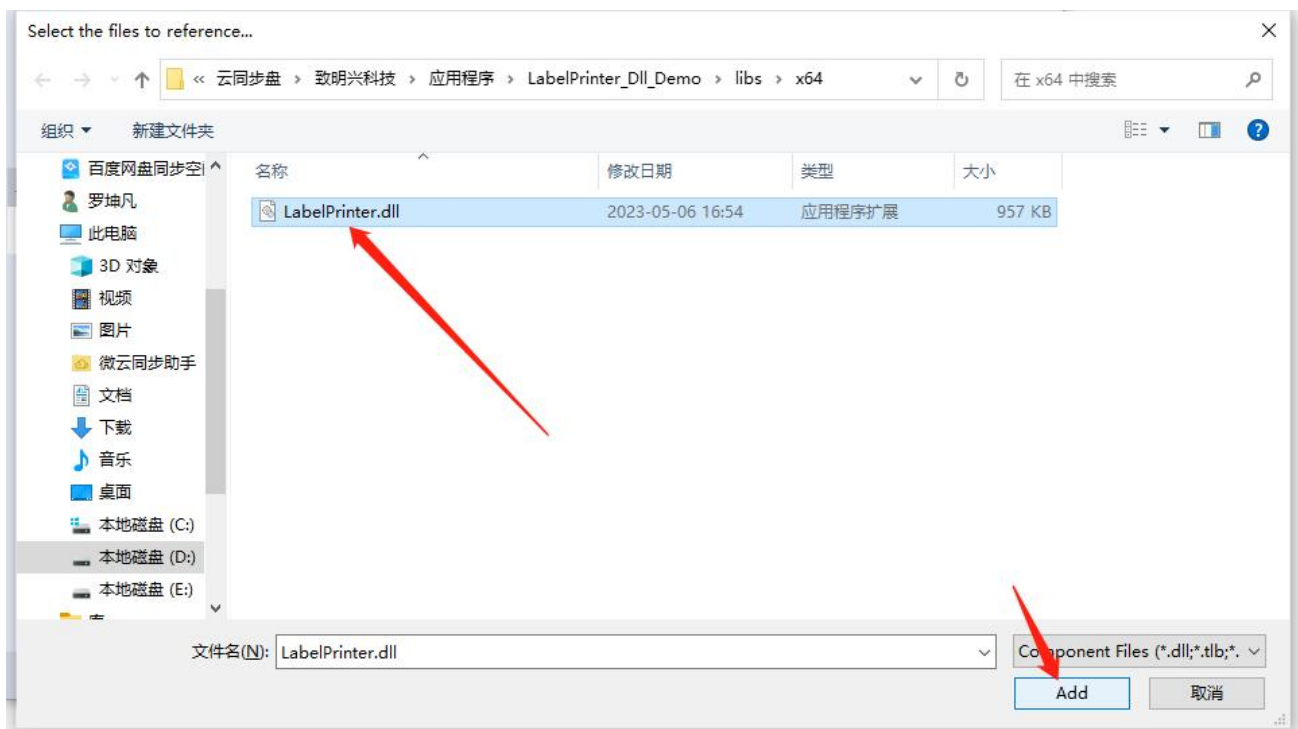
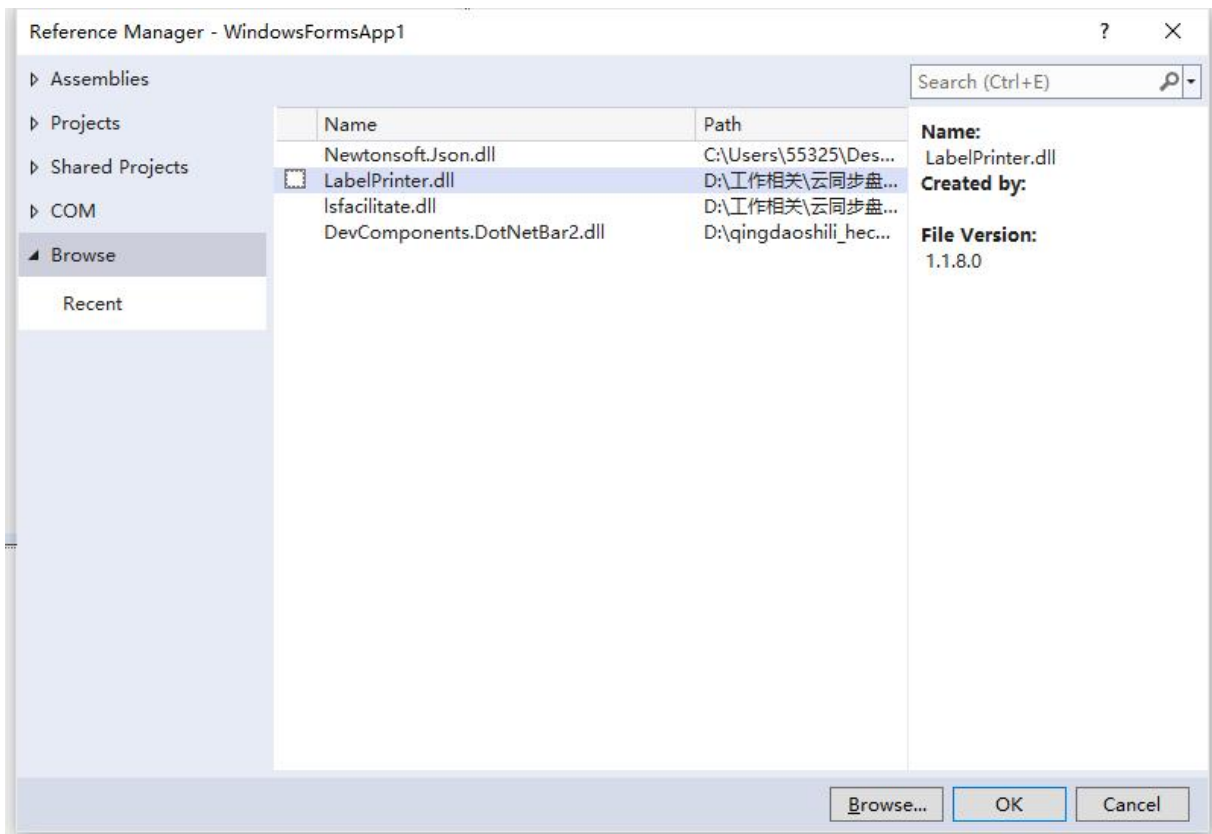
(VS2022 C# .Net Framework):

1. How to reference a DLL file

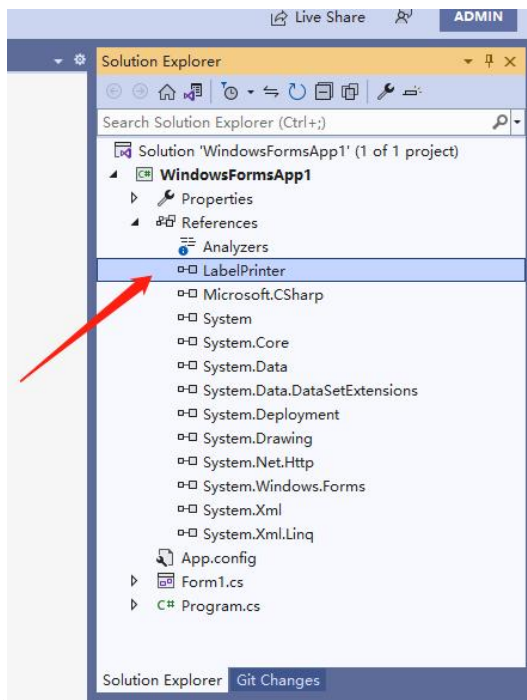
(1) Right click on the reference in the project and select 'Add Reference' from the pop-up menu.



(2) In the pop-up Reference Manager, click the 'Browse' button and select the LabelPrinter.dll file.



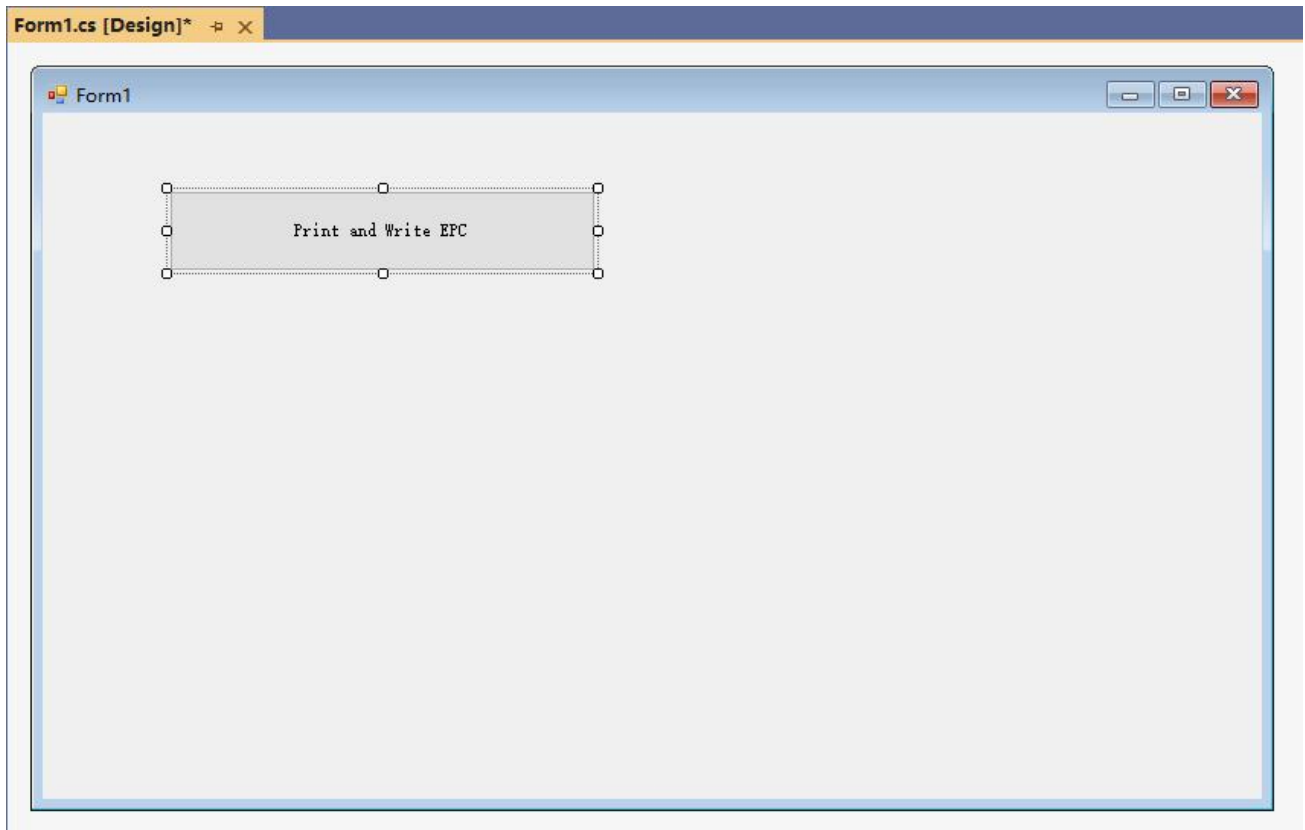
(3) Clicking on the reference in Solution Explorer will reveal that the LabelPrinter has been successfully



added.

2. Writing Code

Create a new form and add a button, taking the "Print and Write EPC" of an UHF printer as an example;



```
LabelPrinter.ZMPrinter thiszmprinter = new LabelPrinter.ZMPrinter();//Printer Object  
LabelPrinter.ZMLabel thiszmlabel = new LabelPrinter.ZMLabel();//label object  
List<LabelPrinter.LabelObject> thiszmlabelobjectlist = new List<LabelPrinter.LabelObject>();//List of objects within  
the label
```

```
1 reference  
private void button1_Click(object sender, EventArgs e)  
{  
    LabelPrinter.ZMPrinter thiszmprinter = new LabelPrinter.ZMPrinter();//Printer Object  
    LabelPrinter.ZMLabel thiszmlabel = new LabelPrinter.ZMLabel();//label object  
    List<LabelPrinter.LabelObject> thiszmlabelobjectlist = new List<LabelPrinter.LabelObject>();//List of objects within t  
}
```

The main code is divided into four parts: defining a printer, defining labels, defining various objects within labels, and printing labels.

(1) Define printer

LabelPrinter.dll defines a printer information class: ZMPrinter

The main job is to declare the type of printer (i.e. data transmission interface), resolution, printing speed, printing blackness, IP address, name, and other information.

The data transmission interface is an enumeration type: PrinterStyle, which has the following types of transmission interfaces:

```

public enum PrinterStyle
{
    DRIVER, //with a value of 0, to print a regular label through the driver, the printer name printername needs
to be defined
    USB, //value is 1, (recommended) Print a regular label through the USB port (no driver installation required)
    NET, //value is 2, printing regular labels through network IP (no driver installation required)
    RFID_DRIVER, //with a value of 3, print RFID tags through the driver program
    RFID_USB, //value is 4, (recommended) Print RFID tags through the USB port (no driver installation required)
    RFID_NET, //value is 5, printing RFID tags through network IP (no driver installation required)
}

```

If the computer is directly connected to the printer through a USB data cable, we recommend using USB or RFID_USB transmission interface, no need to install drivers, simple and convenient to use.

Example:

```

//Define printer
LabelPrinter.ZMPrinter thiszmprinter = new LabelPrinter.ZMPrinter();//Printer Object
thiszmprinter.printerinterface = LabelPrinter.PrinterStyle.RFID_USB;
//thiszmprinter.printername = "RFID Printer";//Printer driver name, valid only when interface type is DRIVER or
RFID_DRIVER
//thiszmprinter.printernetip = "192.168.1.180";//The IP address of the printer, valid only when interface type is
NET or RFID_NET
thiszmprinter.printerdpi = 300;//Printer resolution
thiszmprinter.printSpeed = 4;//Printing speed in IPS
thiszmprinter.printDarkness = 10;//Print darkness, value range: 0-20
thiszmprinter.labelhavegap = true;//Are there gaps, black markings, or holes in the labels.

```

(2) Define labels

LabelPrinter.dll defines a label information class: ZMLabel

The main task is to determine the size of the label to be printed.

The attribute labelwidth is the width of the label;

The attribute labelheight is the height of the label;

The attribute labelrowgap is the row gap of the label, measured in millimeters (mm).

Define a 78X48mm label with a 2mm gap code example:

```

//Define label
LabelPrinter.ZMLabel thiszmlabel = new LabelPrinter.ZMLabel();//label object
thiszmlabel.labelwidth = 78;//The width of the label, in millimeters
thiszmlabel.labelheight = 48;//The height of the label, in millimeters
thiszmlabel.labelrowgap = 2;//The row gap of the label, in millimeters

```

(3) Define each object within the label

LabelPrinter.dll defines an object information class that needs to be printed within a label: LabelObject

The main task is to determine the object name, data, font, font size, barcode type, and other content that needs

to be printed.

There are various objects that need to be printed inside the label: text, one-dimensional barcode, two-dimensional barcode, image, straight line, slash, rectangle, etc.

First, define a list of objects, and then define individual objects separately. After setting the various properties of the object, add the individual object to the object list:

```
List<LabelPrinter.LabelObject> thiszmlabelobjectlist = new List<LabelPrinter.LabelObject>();
LabelPrinter.LabelObject textobject01 = new LabelPrinter.LabelObject();
textobject01.ObjectName = "text-01";// Declare that the object is text.
textobject01.objectdata = "This is paragraph text, containing some characters. ";// Content of text
textobject01.Xposition = 25.1f;// X coordinate of the starting point of the text, in mm
textobject01.Yposition = 35.5f;// Y coordinate of the starting point of the text, in mm
textobject01.texttextalign = 0;// 0 is left aligned, 1 is centered, and 2 is right aligned
textobject01.texttextvalign = 0;// 0 for top edge alignment, 1 for vertical center, and 2 for bottom
edge alignment
textobject01.texttype = 1;// Text type, 0 is single line text, 1 is paragraph text, and 2 is circular
wrapped text
textobject01.textwidth = 50f;// The width of the paragraph text, in millimeters
textobject01.textwidthbeyond = 0;// After exceeding the width of the paragraph text, 0 is for
automatic line wrapping, 1 is for automatic flattening of the font, and 2 is for automatic font reduction
textobject01.textfont = "Arial";// Font of text
textobject01.fontstyle = 1;// 0 regular, 1 bold
textobject01.fontsize = 8;// Font size
thiszmlabelobjectlist.Add(textobject01);// Add a text object to the object list

LabelPrinter.LabelObject textobject02 = new LabelPrinter.LabelObject();
textobject02.ObjectName = "text-02";// Declare that the object is text.
textobject02.objectdata = "This is a single line text. ";// 文字
textobject02.Xposition = 5;// X coordinate of the starting point of the text, in mm
textobject02.Yposition = 6;// Y coordinate of the starting point of the text, in mm
textobject02.textfont = " Arial ";// Font of text
textobject02.fontstyle = 0;// 0 regular, 1 bold
textobject02.fontsize = 10;// Font size
thiszmlabelobjectlist.Add(textobject02);// Add a text object to the object list

LabelPrinter.LabelObject shape01 = new LabelPrinter.LabelObject();
shape01.ObjectName = "rectangle-01";// Declare that the object is a rectangle.
shape01.objectclass = 2;// The category of objects, 1 is a straight line and 2 is a rectangle
shape01.startXposition = 24f;// The X-coordinate of the starting point, in millimeters
shape01.startYposition = 25f;// The Y-coordinate of the starting point, in millimeters
shape01.endXposition = 76f;// The X-coordinate of the termination point, in millimeters
shape01.endYposition = 45f;// The Y-coordinate of the termination point, in millimeters
shape01.lineWidth = 0.4f;// The width of the line, in millimeters
thiszmlabelobjectlist.Add(shape01); // Add a rectangle object to the object list
```

```

LabelPrinter.LabelObject barcodeobject01 = new LabelPrinter.LabelObject();
barcodeobject01.ObjectName = "barcode-01";// The object is a barcode
barcodeobject01.objectdata = "12345678";// Content of barcode
barcodeobject01.Xposition = 25;// X coordinate of the starting point of the text, in mm
barcodeobject01.Yposition = 12;// Y coordinate of the starting point of the text, in mm
barcodeobject01.barcodekind = "Code 128 Auto";// Type of barcode: Code 128 Auto, Code 128 A, Code 128
B, Code 128 C, EAN-13, QR Code(2D), PDF 417(2D), Code 39, Data Matrix(2D)
barcodeobject01.barcodescale = 3;// Horizontal scaling coefficient of barcode, value range: 1-99, the
larger the number, the wider the barcode
barcodeobject01.barcodeheight = 5;// Barcode height in mm
this.szmlabelobjectlist.Add(barcodeobject01);// Add barcode objects to the object list

LabelPrinter.LabelObject rfidobject01 = new LabelPrinter.LabelObject();
rfidobject01.ObjectName = "rfiduhf-01";// The object is written using RFID
rfidobject01.objectdata = "12345678";// The content that needs to be written in RFID is recommended
to be a multiple of 8 in length.
rfidobject01.RFIDatablock = 0;// Write area, 0 is EPC, 1 is USER
rfidobject01.RFIDdatatype = 1;// Data type: 0 is text, 1 is hexadecimal
rfidobject01.RFIDerrortimes = 1;// Error retry count
this.szmlabelobjectlist.Add(rfidobject01);// Add RFID object to object list

```

If multiple text or other content needs to be printed, different individual objects need to be defined and added to the object list, as shown in the example above.

The object property **ObjectName** declares the object type, and its value has a specific meaning and cannot be set arbitrarily. The specific meaning is as follows:

- ① **barcode** object is a barcode, can be set to "barcode-01", "barcode-02", and so on. which requires setting the position of the upper left corner, the type of barcode, scaling ratio, barcode height, barcode content, etc.
- ② **text** object is TrueType text, can be set to "text-01", "text-02", and so on. which requires setting the top left corner position, font name, font size, text content, and more.
- ③ **line** object is a straight line and requires setting the starting point coordinates, line width, style, etc.
- ④ **rectangle** object is a rectangle that requires setting the starting point coordinates, line width, style, and so on.
- ⑤ **image** object is an image, which requires setting the position of the top left corner, image name, image binary data, horizontal and vertical scaling ratios, etc.
- ⑥ **rfiduhf** object is an RFID write object that requires setting the write area, content, format, etc.

The attribute **barcodekind** is the type of barcode, which is a string.

The following barcode types are supported:

Code 128 Auto

Code 128 A

Code 128 B

Code 128 C

EAN-13

QR Code(2D)

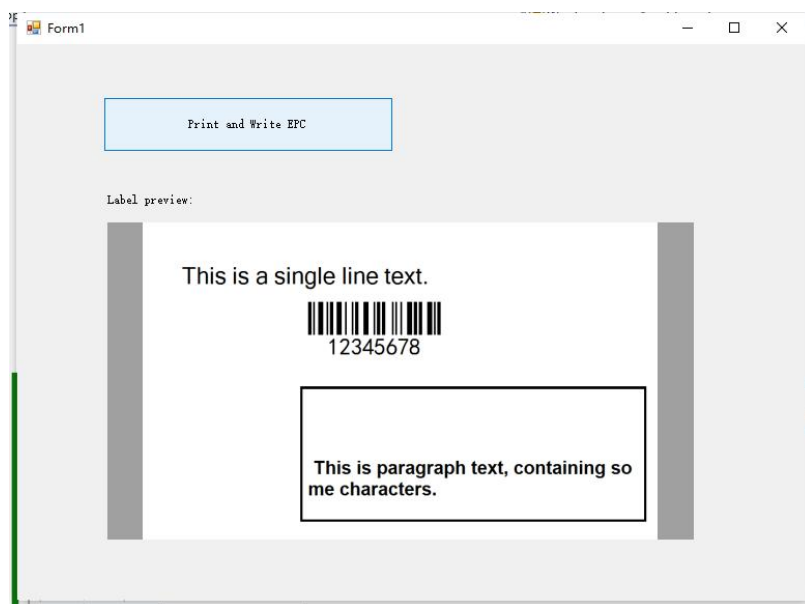
PDF 417(2D)

Code 39

Data Matrix(2D)

(4) Preview Label

```
#region Label preview
    Bitmap previewimage = new LabelPrinter.PrintUtility().GetLabelImage(thiszmpainter, thiszmlabel,
thiszmlabelobjectlist, 0); //Generate preview image
    //previewimage.Save("c:\\test.png", System.Drawing.Imaging.ImageFormat.Png);
    if (previewimage != null)
        pictureBox1.Image = previewimage;
#endregion
```



(5) Print labels

```
#region print label
    string returnmsg = new LabelPrinter.PrintUtility().PrintLabel(thiszmpainter, thiszmlabel,
thiszmlabelobjectlist, true, true);
    if (returnmsg.StartsWith("Error:"))
        MessageBox.Show(returnmsg, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
#endregion
```

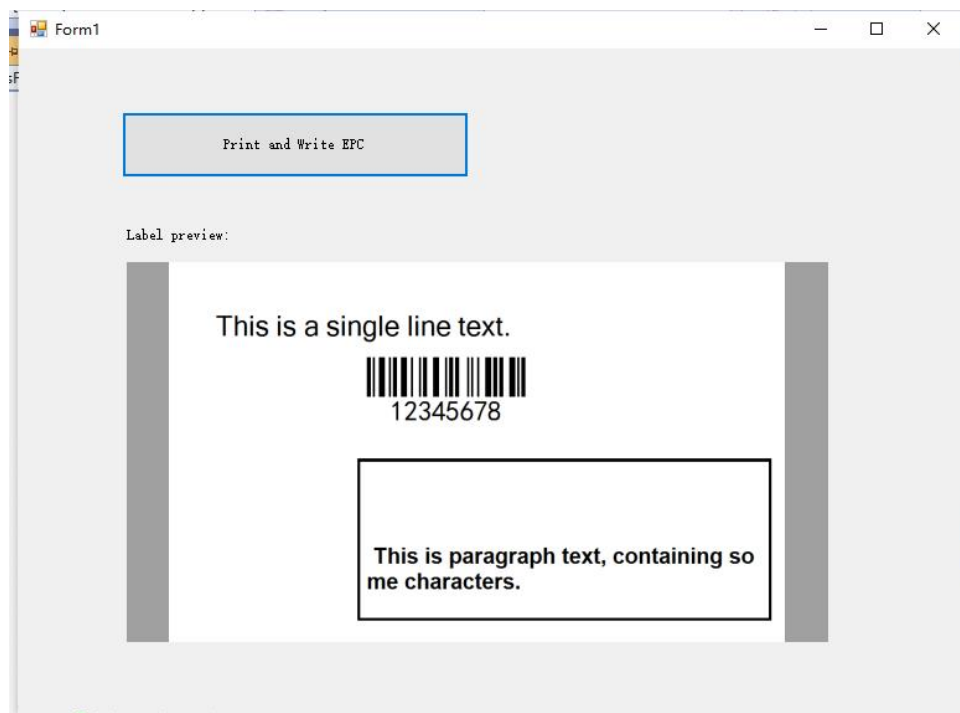
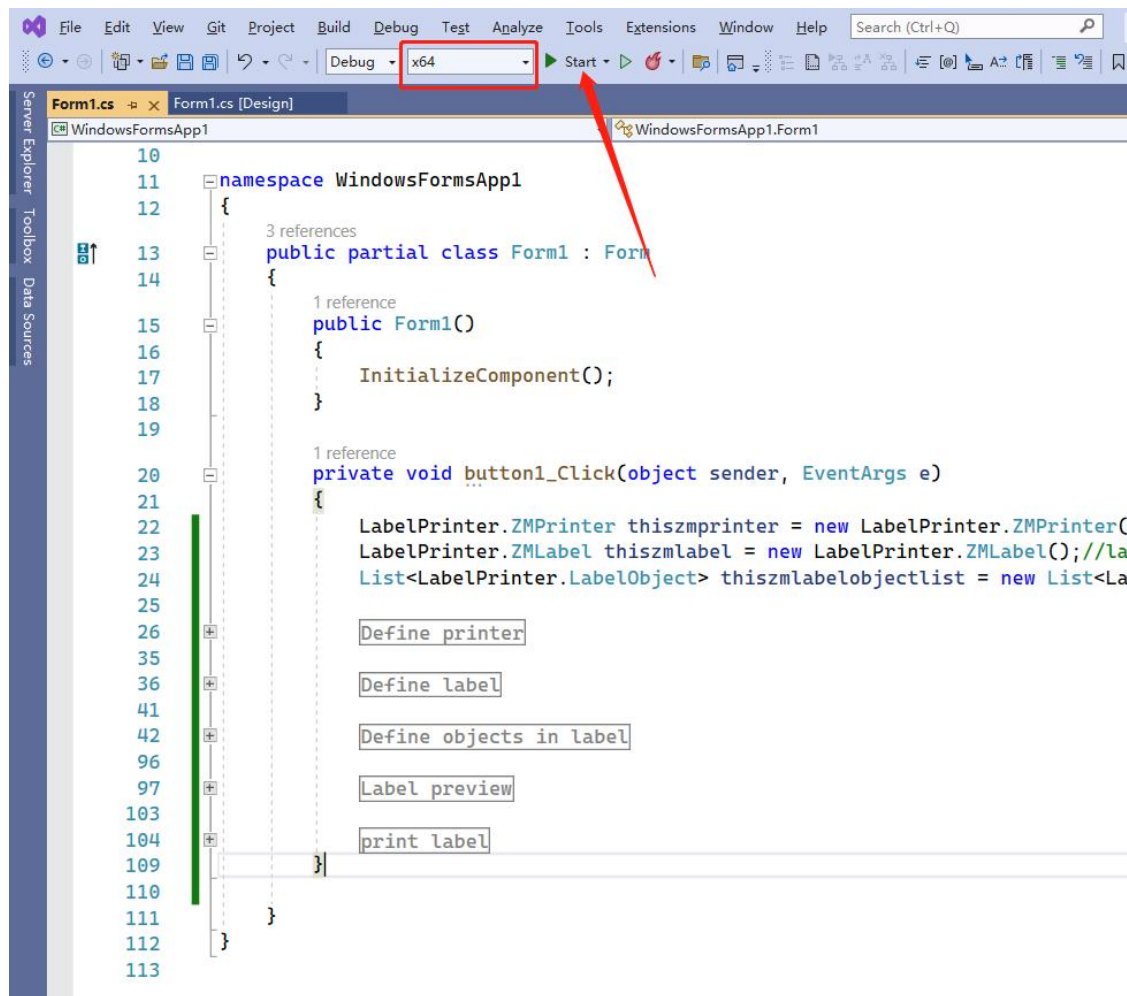
LabelPrinter.dll defines a printing processing class: PrintUtility

The PrintLabel function includes querying the printer status, converting the object list into print data, and sending the data to the printer.

3. Run Code

After the code is written, it can be run and debugged in VS.

Note that the platform should be set to be the same as the referenced LabelPrinter.dll.



4. Introduction to Other Functions

(1) Read LSF label template

Function prototype: string LSFUtility.OpenLabel(string filename, ref ZMPrinter thisprinterinfo, ref ZMLabel plabel, ref List<LabelObject> LabelObjectList)

Function: Read LSF label documents to generate printer, label, and content information required for printing.

Input parameters:

- ① Lsf label file name (full path);
- ② Printer object: ZMPrinter;
- ③ Label object: ZMLabel;
- ④ List of objects within the label (text, barcode, image, etc.): List<LabelObject>.

Function returns: string

If read fails, an error message will be returned. Read successfully will return an empty string.

Example:

#region Read LSF document

```
string openstatus = new LabelPrinter.LSFUtility().OpenLabel(labelfilepath, ref thiszmprinter, ref thiszmlabel, ref thiszmlabelobjectlist); // Read LSF document
if (!string.IsNullOrEmpty(openstatus)) // Read error
{
    MessageBox.Show(openstatus, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    return;
}
#endregion
```

(2) Reading TID or EPC of UHF tags

Function prototype: string GetUHFTagData (ZMPrinter thiszmprinter, ZMLabel thiszmlabel, int area, int modulePower, int stopposition, int timeout)

Function: Read UHF tag data, support USB and network printers.

Input parameters:

- ① Printer object: ZMPrinter;
- ② Label object: ZMLabel;
- ③ The label area that needs to be operated (0 represents TID, 1 represents EPC, and 2 represents TID+EPC);
- ④ Module reading power (values range from 0 to 25dBm, 0 is not specified, using the power already set by the printer);
- ⑤ After the reading operation is completed, the label stops at the position (0 represents returning to the original position, 1 represents walking to the tearing position, 2 represents walking to the printing position, and 3 represents walking to the writing position (data needs to be written));
- ⑥ Read timeout, in milliseconds (recommended to set to 2000).

Function returns:

Read normally, return the data read;

Read an exception and return an empty string or "Error:"+error message.

Example:

#region Reading the TID of UHF tags

```
/*GetUHFTagData(ZMPrinter,ZMLabel,0,0,3,2000);
```

Parameter Description:

Parameter 1 is the printer object ZMPrinter;

Parameter 2 is the label object ZMLabel;

Parameter 3 is the label area that needs to be operated on (0 is TID, 1 is EPC, and 2 is TID+EPC);

Parameter 4 is the module reading power (values are 0-25dBm, 0 is not specified, using the power already set by the printer);

Parameter 5 is the stop position of the label after the reading operation is completed (0 is to return to the original position, 1 is to walk to the tearing position, 2 is to walk to the printing position, and 3 is to walk to the writing position (data needs to be written));

Parameter 6 is the read timeout time in milliseconds (recommended setting is 2000).

Return value:

1. Reading normal, returning the data read

2. Read an exception and return an empty string or "Error:"+error message

```
*/
```

```
string tagdata = new LabelPrinter.PrintUtility().GetUHFTagData(thiszmprinter, thiszmlabel, 0, 0, 2, 2000); //Read
```

the TID of the UHF tag, and only print it later without writing data

```
if (string.IsNullOrEmpty(tagdata) || tagdata.StartsWith("Error:"))
```

```
{
```

```
    /*PrintaBlankpage(ZMPrinter,ZMLabel,1);
```

Parameter Description:

Parameter 1 is the printer object ZMPrinter;

Parameter 2 is the label object ZMLabel;

Parameter 3 is whether to print error identifier X (0: do not print, 1: print X);

Return value:

No return value

```
*/
```

```
new LabelPrinter.PrintUtility().PrintaBlankpage(thiszmprinter, thiszmlabel, 1); // Print a label with incorrect  
identification
```

```
MessageBox.Show("No label data was read.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
return;
```

```
}
```

```
#endregion
```

(3) Read the UID of high-frequency tags

Function prototype: string GetHFTagData (ZMPrinter thiszmprinter, ZMLabel thiszmlabel, int protocol, int area, int modulePower, int stopposition, int timeout)

Function: Read high-frequency tag data, support USB and network printers.

Input parameters:

① Printer object: ZMPrinter;

② Label object: ZMLabel;

③ High frequency protocol type (1 is 15693, 2 is 14443A, and 3 is NFC);

④ The label area that needs to be operated (0 is the UID, 1 is the data area);

⑤ Module reading power (0:12db, 1:24db, 2:36db, 3:48db);

- ⑥ After printing this label, locate whether the next label stops at the paper tearing port; 0: Below the print head, 1: Paper tearing port;
- ⑦ Read timeout, in milliseconds (recommended to set to 2000).

Function returns:

Read normally, return the data read;

Read an exception and return an empty string or "Error:"+error message.

Example:

#region Read the UID of HF tags

```
/* GetHFTagData(ZMPrinter, ZMLabel, 1, 0, 0, 1, 2000);
```

Parameter Description:

Parameter 1 is the printer object ZMPrinter;

Parameter 2 is the label object ZMLabel;

Parameter 3 is the high-frequency protocol type (1 is 15693, 2 is 14443A, and 3 is NFC);

Parameter 4 is the label area that needs to be operated on (0 is UID, 1 is data area);

Parameter 5 is the module reading power (0:12db, 1:24db, 2:36db, 3:48db);

Parameter 6 determines whether the next label will stop at the paper tearing port after printing this label;

0: Below the print head, 1: Paper tearing port

Parameter 7 is the read timeout time in milliseconds (recommended setting is 2000).

Return value:

1. Reading normal, returning the data read

2. Read an exception and return an empty string or "Error:"+error message

```
*/
```

```
string tagdata = new LabelPrinter.PrintUtility().GetHFTagData(thiszmprinter, thiszmlabel, 1, 0, 0, 1, 2000); // Read the UID of high-frequency tags, and only print without writing data afterwards.
```

```
if (string.IsNullOrEmpty(tagdata) || tagdata.StartsWith("Error:"))
```

```
{
```

```
    /*PrintaBlankpage(ZMPrinter,ZMLabel,1);
```

Parameter Description:

Parameter 1 is the printer object ZMPrinter;

Parameter 2 is the label object ZMLabel;

Parameter 3 is whether to print error identifier X (0: do not print, 1: print X);

Return value:

No return value

```
*/
```

```
new LabelPrinter.PrintUtility().PrintaBlankpage(thiszmprinter, thiszmlabel, 1); // Print a label with incorrect identification
```

```
MessageBox.Show("No label data was read.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
return;
```

```
}
```

#endregion

(4) Print a blank page

Function prototype: void PrintaBlankpage (ZMPrinter thiszmprinter, ZMLabel thiszmlabel, int printerrorflag)

Function: After reading an RFID tag fails, you need to exit the tag and set whether to print the error label.

Input parameters:

- ① Printer object: ZMPrinter;
- ② Label object: ZMLabel;
- ③ Whether to print error identification X (0: do not print, 1: print X).

Function returns:

None.

(5) Query printer status function

Function prototype: int getPrinterStatusCode (ZMPrinter thiszmprinter)

Function: Read the current status of the printer.

Input parameters:

- ① Printer object: ZMPrinter;

Function returns:

The status code of the int printer.

(6) Query Printer Status Function

Function Prototype: int getPrinterStatusCode(ZMPrinter thiszmprinter)

Description: Retrieves the current status of the printer.

Input Parameters:

- ① Printer object: ZMPrinter.

Return Value:

int The status code of the printer.

(7) Function Prototype: string PrinterErrorCodeToString(int statuscode)

Description: Converts the error code into a Chinese error message.

Input Parameters: The return value of getPrinterStatusCode.

Return Value: string errormsg.

Note: It is recommended to call this function only when the status code is non-zero.

Example:

```
for (int i = 0; i < 3; i++)
{
    int statuscode = pUtility.getPrinterStatusCode(thiszmprinter); //查询打印机状态 check the printer status

    if (statuscode != 0)
    {
        //错误处理 error handling
        string errorinfo = pUtility.PrinterErrorCodeToString(statuscode);
        MessageBox.Show(errorinfo, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        break;
    }
}
```

(8) Send Printer Commands

Function Prototype: `string SetPrinterParams(ZMPrinter thiszmprinter, string paramstring)`

Description: Sends control commands to the printer, which can be used to configure the printer or print labels.

Input Parameters:

- ① Printer object: ZMPrinter;
- ② Command string.

Return Value:

string The status of the printer.

Example:

```
LabelPrinter.ZMPrinter thiszmprinter = new LabelPrinter.ZMPrinter();//Printer Obeject
thiszmprinter.printerinterface = LabelPrinter.PrinterStyle.USB;
thiszmprinter.printerdpi = 300;//Printer DPI
thiszmprinter.printSpeed = 4;//Printer Speed
thiszmprinter.printDarkness = 10;//Pritner Drakness
thiszmprinter.labelhavegap = true;//Is there a gap between labels?

string
cmdstring="I8,1,001\r\nS40\r\nH10\r\nq614\r\nQ354,0\r\nN\r\nZB\r\nT100,71,0,3,1,1,N,\"Test012345\"\r\n
W1,1";//Print Code
LabelPrinter.PrintUtility printUtility = new LabelPrinter.PrintUtility();//Print Handler Class
string returnmsg = printUtility.SetPrinterParams(thiszmprinter, cmdstring);//Send code
if (returnmsg.StartsWith("Error:"))
{
    MessageBox.Show(returnmsg, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

This document was last edited on May 07, 2026.

Introduction to the parameters of printer class, label class, and various object classes that need to be printed within the label:

Detailed parameter list of ZMPrinter class:

```
PrinterStyle printerinterface = PrinterStyle.USB;// The interface type of the printer
string printername="Default";// Printer Driver Name
int printerdpi=300;// Printer resolution, 203, 300, or 600dpi
int printSpeed=4;// Printing speed, in inches per second, effective from 2 to 8
int printDarkness=10;// Printing blackness, effective from 0 to 20, with higher values leading to higher printing temperature
string printernetip="192.168.8.180";// The IP address of the printer
bool labelhavegap = true;// Is there a gap between the labels.
int pageDirection = 1;// 1 vertical, 2 horizontal
bool reverse = false;// Whether to rotate 180 degrees for printing, i.e. reverse printing
```

Detailed parameter list for ZMLabel class:

```
float labelwidth=60;// Label width in mm
float labelheight=40;// Label height in mm
float labelrowgap=2;// Line spacing in mm
float labelcolumngap=2;// Column spacing, in mm
int labelrownum=1;// Number of rows
int labelcolumnnum=1;// Number of columns
float leftoffset=0;// Fine adjustment of left position in mm
float topoffset=0;// Fine adjustment of top position in mm
float pageleftedges=0;// Left space, in mm
float pagerightedges=0;// Right space, in mm
int pagestartlocation=0;// Starting position, 0 is top left, 1 is top right, 2 is bottom left, and 3 is bottom right
int pagelabelorder=0;// Label order, 0 is horizontal, 1 is vertical
int labelshape = 0;// The shape of the label, 0 rounded rectangle, 1 square corner rectangle, 2 ellipse
```

Detailed parameter list of LabelObject classes that need to be printed in the label:

```
string ObjectName="";// The name of the label object
//Naming rules for object names:
//1. Barcode objects start with "barcode", such as "barcode-01", "barcode-02"
//2. Text objects start with "text", such as "text-01", "text-02"
//3. Line objects start with "line", such as "line-01", "line-02"
//4. A rectangular object starts with 'rectangle', such as 'rectangle-01 ', 'rectangle-02 '
//5. The image object starts with "image", such as "image-01", "image-02"
//6. RFID objects start with "rfiduhf", such as "rfiduhf-01", "rfiduhf-02"
//Note: The names of objects in both UHF and HF start with "rfiduhf"
```

```
string objectdata="";// The content of the object
float Xposition=3;// The position on the label, in millimeters
float Yposition=3;// The position on the label, in millimeters
float startXposition=3;// The position of the starting point of the line and rectangle on the label, in millimeters
```

```

float startYposition=3;// The position of the starting point of the line and rectangle on the label, in millimeters
float endXposition=10;// The position of the endpoint of the line and rectangle on the label, in millimeters
float endYposition=10;// The position of the endpoint of the line and rectangle on the label, in millimeters
float lineWidth=0.4f;// The width of straight and rectangular lines, in millimeters
int lineDashStyle=0;// line style, 0 is a solid line, 1 is a broken dashed line, 2 is a broken dotted line, 3 is a broken dotted line, and 4 is a dotted line
bool fillRectangle=false;// Whether to fill the rectangle
int lineclass = 1;// Category of straight lines, 1 is a horizontal line, 2 is a vertical line, and 3 is a diagonal line
int rectangleclass=0;// The category of rectangles, where 0 is a right angle rectangle

string barcodekind = "Code 128 Auto";// Barcode type, refer to the barcode type in LabelSoft software
int barcodescale=3;// Barcode scaling factor, integer, usually ranging from 1 to 99
float barcodeheight=5;// The height of a one-dimensional barcode, in millimeters, does not need to be set for two-dimensional barcodes
int direction=0;// The direction of rotation for barcodes or text, where 0 is 0 degrees, 1 is 90 degrees, 2 is 180 degrees, and 3 is 270 degrees
int errorcorrection=0;// QR code error correction level, 0: L, 1: M, 2: Q, 3: H
int charencoding=0;// Character encoding of QR code: 0 is UTF-8, 1 is GB2312
int qrversion = 0;// QR symbol version, 1-40, with larger numbers containing more characters, 0 being automatic
int code39widthratio=3;// Bar width ratio of Code39
bool code39startchar=true;// Does Code39 contain a start character*
int barcodealign = 0;// The alignment of barcodes, where 0 is left aligned, 1 is center aligned, and 2 is right aligned
int pdf417_rows=0;// The number of rows or layers of PDF417 barcode, where 0 is automatic
int pdf417_columns=0;// The number of columns in the PDF417 barcode, that is, the number of data blocks, excluding the left and right identification blocks. 0 is automatic
int pdf417_rows_ auto = 3;// When the number of lines in the PDF417 barcode is set to automatic, the minimum number of lines obtained is 3
int pdf417_columns_ auto = 1;// When the number of columns in the PDF417 barcode is set to automatic, the minimum number of columns obtained is 1
int datamatrixShape = 0;// The shape of the DataMatrix, where 0 is automatic, 1 is square, and 2 is rectangular

int textposition=0;// The position of the text relative to the barcode, where 0 is below, 1 is above, and 2 is not displayed
float textoffset=0;// The distance between text and barcode, in millimeters
int textalign=2;// The alignment of text relative to the barcode, where 0 is left, 1 is right, 2 is centered, and 3 is full
String textfont="Arial">// Font Name
int fontstyle=0;// Font Style, 0 Normal, 1 Bold, 2 Italic, 3 Italic Bold
float fontsize=8;// font size
bool blackbackground=false;// Is it black background with white characters
float chargap=0;// Character spacing in mm
float charHZoom=1;// Character horizontal scaling factor, 0.3-2
int texttype=0;// Text type, 0 is single line text, 1 is paragraph text, and 2 is circular wrapped text
int texttextalign = 0;// The alignment of text, with 0 being left aligned, 1 being centered, and 2 being right aligned
int texttextvalign = 0;// The alignment of the text, 0 for top edge alignment, 1 for vertical center, and 2 for bottom edge alignment

```

```

float textwidth=30;// The width of the paragraph text, in millimeters
int textwidthbeyond = 0;// After the paragraph text exceeds the specified width, it will be processed. 0 is for
automatic line wrapping, 1 is for automatic flattening of the font, and 2 is for automatic font reduction
int linegapindex=0;// Line spacing of paragraph text, 0 is single, 1 is one and a half times, 2 is double, and 3 is
custom
float linegap=0;// Custom line spacing
float circularradius = 2f;// The radius of a circle surrounding text
int textradian = 360;// Text radian around circular text
int textstartangle = 0;// The starting angle of circular wrapping text
int rewindingdirection = 0;// The wrapping direction of circular wrapping text, 0 clockwise, 1 counterclockwise
int literaldirection = 0;// The text direction of the circular wrapping text, 0 outward and 1 inward

byte[] imagedata;// Data for storing images
bool aspectRatio = true;// Does the image maintain aspect ratio
float hscale = 1;// Horizontal scaling percentage
float vscale = 1;// Vertical scaling percentage
bool imagefixedsize = false;// Fixed size or not
float imagefixedwidth = 0;// Fixed width of the image, in millimeters
float imagefixedheight = 0;// Fixed height of the image, in millimeters

bool transparent = true;// Is the background transparent

//The following are RFID parameters
int RFIDEncodertype = 0;// 0 is UHF, 1 is HF 15693, 2 is HF 14443A, and 3 is NFC

//UHF
int RFIDDatablock = 0;// Write data area: 0 is EPC, 1 is USER
int RFIDDatatype = 1;// Data type written: 0 is text, 1 is hexadecimal
int RFIDTextencoding = 0;// Text encoding: 0 is ASCII, 1 is UTF-8
int DataAlignment = 0;// Data alignment method, 0 represents backend zero padding, and 1 represents front-end
zero padding
int RFIDErrortimes = 2;// retry count
bool Datalengthdoublewords=false;// Force writing in units of 4 bytes

//HF
int HFstartblock = 0;// High frequency, the starting address of the block to be written
int HFmodulepower = 0;// High frequency module power
bool Encrypt14443A = false;// Do you want to encrypt the 14443A label
int Sector14443A = 1;// Sectors of 14443A that require encryption
int KEYAB14443A = 0;// KEYA or KEYB encryption is required, 0 is KEYA, 1 is KEYB, and 2 is both encrypted
string KEYAnewpwd = "";// KEYA New Password
string KEYAoldpwd = "FFFFFFFFFFFF";// KEYA Old Password
string KEYBnewpwd = "";// KEYB New Password
string KEYBoldpwd = "FFFFFFFFFFFF";// KEYB Old Password
bool Encrypt14443AControl = false;// Do you want to set the control word for 14443A

```

```
string Encrypt14443AControlvalue = "FF078069";// Default Control Word  
int Controlarea15693 = 0;// 0 is not set, 1 is AFI, and 2 is DSFID  
string Controlvalue15693 = "00";// The set value defaults to 00
```

End of document.