Zebra® ZXP Series 1 & ZXP Series 3™

Software Developers Reference Manual (SDK)

Requires Printer Driver

ZEBRA TECHNOLOGIES

P1035767-001 (Draft 5)

Copyright Notice

Trademarks

P1004475 Rev. A

Contents

1 Introduction

About This Manual

This manual contains information for software developers intending to write applications for Zebra ZXP Series 1 and ZXP Series 3 Card Printers. The application programming interface (API) provides functions to access card printer features. The Zebra printer drivers run on the following Windows Operating Systems:

- Windows XP
- Windows Server 2003 and Server 2008
- Windows Vista
- Windows 7 (32/64 bits)

This manual is part of the Zebra Card Printer Software Developer's Kit (SDK).

Required Skills

- Experience in developing applications for the Microsoft Windows environment
- Experience in developing applications using dynamic link libraries (dll)
- Experience with Microsoft's Windows Graphics Device Interface (GDI)

Zebra Card Printers

This manual describes the programming functions that control operations and deliver data for Zebra ZXP Series 1 and ZXP Series 3 Card Printers.

Communication Ports

- USB 2.0
- Ethernet

SDK Elements

Printer

- ZBRPrinter.dll
  - 32 bit dynamic link library
  - calling convention is __stdcall
- ZBRPrinter.h
- C# sample code

Graphics

- ZBRGraphics.dll
  - 32 bit dynamic link library
  - calling convention is __stdcall
- ZBRGraphics.h
- C# sample code

SCEncoding Ethernet Interface

- ZBRSXBridge.dll
  - 32 bit dynamic link library
  - calling convention is __stdcall
- C# sample code

Installation

Directory Structure

(Disk Drive):\Zebra SDK\Printer\ #.##.## \doc
\bin
\sample

(Disk Drive):\Zebra SDK\Graphics\ #.##.## \doc
\bin
\sample

doc directory contains SDK documentation

bin directory contains the dynamic link library (dll) and include files

sample directory contains example applications

System Directories

SDK dll files should be placed in the system directory.

Example -- XP

(Disk Drive):\WINDOWS\system32\

Card-Handling

In the following card-handling sequence, do encoding first (SmartCard encoding before the Magnetic Stripe encoding); then do card printing:

1. Feed Card (manual or auto) into printer
2. Clean Card
3. Encode Card -- SmartCard Option
4. Encode Card -- Magnetic Stripe Option
5. Print Card (front side)

   For color, print:

   Yellow
   Magenta
   Cyan
   Black
   Clear Varnish

6. Flip Card
7. Clean Card
8. Print Card (back side)

   For color, print:

   Yellow
   Magenta
   Cyan
   Black
   Clear Varnish
   Hologram Lamination

9. Eject Card

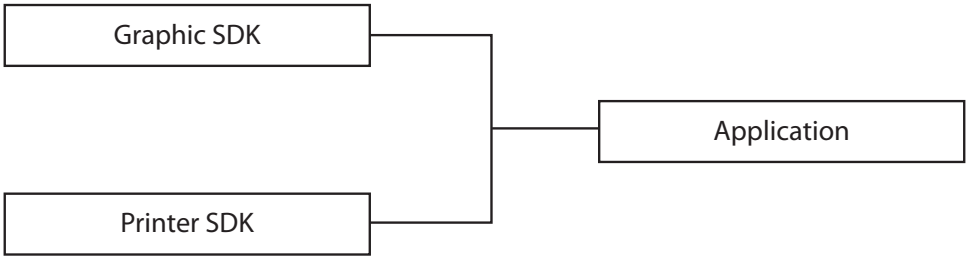
2

Software Development Kit

2.1 Introduction

The ZXP Series 1 and ZXP Series 3 Software Development Kit (SDK) provides developers a platform for designing applications for ZXP Series 1 and ZXP Series 3 Printers.

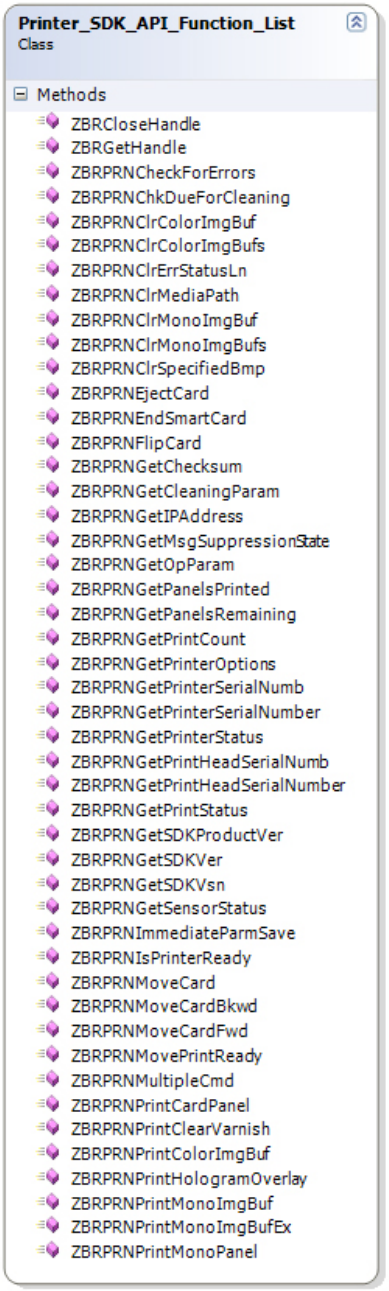
2.2 SDK Diagram

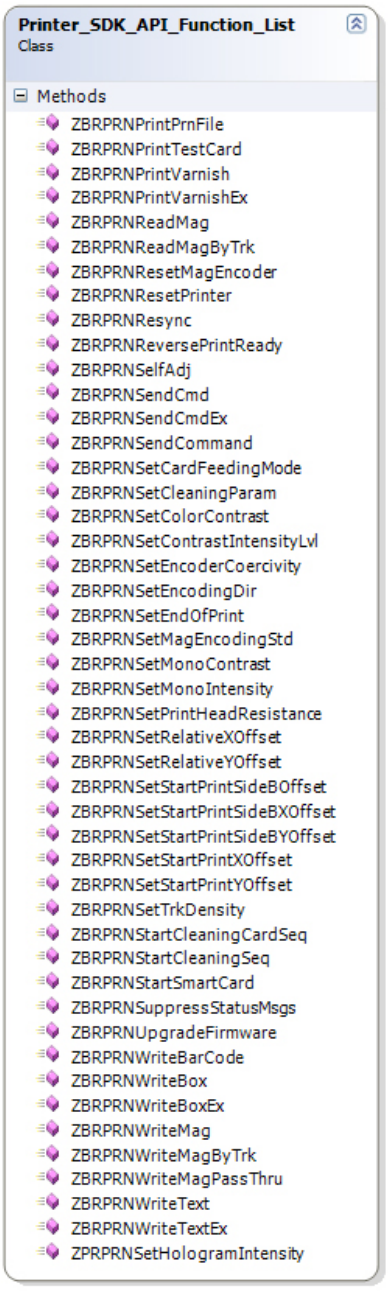


2.3 SDK Diagrams

Printer Function List

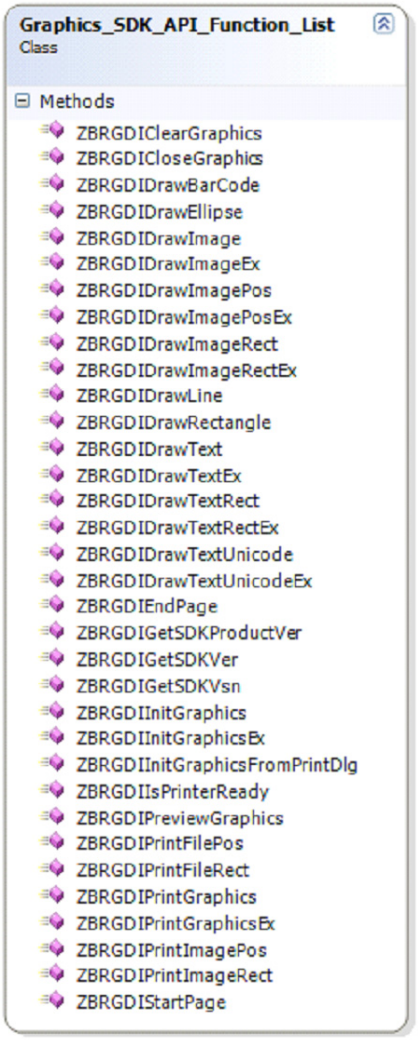
The Printer SDK Diagram depicts the SDK as a class diagram. This was done for ease of illustration purposes. The Printer SDK is not a class; instead, it is a list of functions written in the C programming language and exported through a C language dynamic link library: ZBRPrinter.dll.





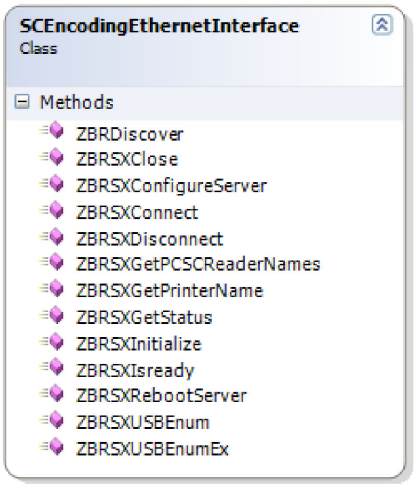
Graphics Function List

The Graphics SDK Diagram depicts the SDK as a class diagram. This was done for ease of illustration purposes. The Graphics SDK is not a class; instead, it is is a list of functions written in the C programming language and exported through a C language dynamic link library: ZBRGraphics.dll.



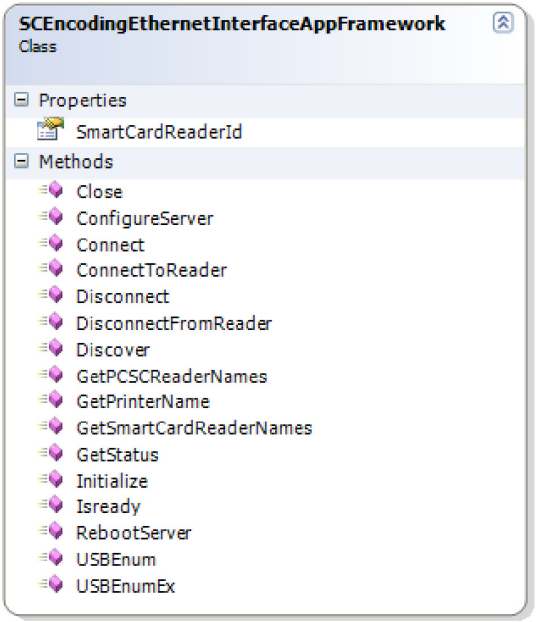
Ethernet SmartCard Encoding SDK

The Ethernet SmartCard Encoding SDK Diagram depicts the SDK as a class diagram. This was done for ease of illustration purposes. The Ethernet SmartCard Encoding SDK is not a class; instead, it is a list of functions written in the C programming language and exported through a C language dynamic link library: ZBRSXBridge.dll.



Ethernet SmartCard Encoding Application Framework

The Ethernet SmartCard Encoding Application Framework is a class written in .NET available as a dynamic link library: ZSCEncodeAP.dll.



3

Printer Functions

Introduction

This section contains information for software developers intending to write applications for Zebra ZXP Series 1 and ZXP Series 3 Card Printers. The Application Programming Interface (API) provides functions to access card printer features.

Required Skills

- Experience in developing applications for the Microsoft Windows environment
- Experience in developing applications using dynamic link libraries (dll)

Zebra Card Printers

- ZXP Series 1
- ZXP Series 3

Communication Ports

- USB 2.0
- Ethernet

Printer SDK Elements

- ZBRPrinter.dll
  - 32 bit dynamic link library
  - calling convention is __stdcall
- ZBRPrinter.h
- C# sample code

Installation

Directory Structure

(Disk Drive):\Zebra SDK\Printer\#.##.## \doc
\bin
\sample

doc directory contains SDK documentation
bin directory contains the dynamic link library (dll) and include files
sample directory contains example applications

System Directories

SDK dll files should be placed in the system directory.

Example:

XP

(Disk Drive):\WINDOWS\system32\

Function List

SDK Specific Function

ZBRPRNGetSDKVer

Description: Returns the SDK dll version.

Syntax:
```
void ZBRPRNGetSDKVer(
                        int           *major,
                        int           *minor,
                        int           *engLevel)
```

Parameters:

major	[out]pointer to major version number
minor	[out]pointer to minor version number
engLevel	[out]pointer to engineering level number

Return Value: None

Sample Code:
```
int major = 0;
int minor = 0;
int engLevel = 0;

//import the function from the dll:
[DllImport("ZBRPrinter.dll", EntryPoint = "ZBRPRNGetSDKVer",
    CharSet = CharSet.Auto, SetLastError = true)]
public static extern void ZBRPRNGetSDKVer(out int major,
                                          out int minor,
                                          out int engLevel);

//use the function:

ZBRPRNGetSDKVer(out major, out minor, out engLevel);
```

ZBRPRNGetSDKVsn

Description: Returns the SDK dll version.

Syntax:
```
void ZBRPRNGetSDKVsn(
                        int           *major,
                        int           *minor,
                        int           *engLevel)
```

Parameters:

major	[out]pointer to major version number
minor	[out]pointer to minor version number
engLevel	[out]pointer to engineering level number

Return Value: None

Sample Code:
```
int major = 0;
int minor = 0;
int engLevel = 0;

//import the function from the dll:
[DllImport("ZBRPrinter.dll", EntryPoint = "ZBRPRNGetSDKVsn",
    CharSet = CharSet.Auto, SetLastError = true)]
public static extern void ZBRPRNGetSDKVsn(out int major,
                                          out int minor,
                                          out int engLevel);

//use the function:

ZBRPRNGetSDKVsn(out major, out minor, out engLevel);
```

ZBRPRNGetSDKProductVer

Description: Returns the SDK's product version.

Syntax: `int ZBRPRNGetSDKProductVer(`
`char *productVersion,`
`int *buffSize)`

Parameters:

productVersion	[out]buffer to hold returned product version
buffSize	[out]size of the returned buffer in bytes

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:

```
byte[] productVersion = new byte[50];
int buffSize = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNGetSDKProductVer",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNGetSDKProductVer(byte[] productVersion,
                                                 out int buffSize);



//use the function:

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();

result = ZBRPRNGetSDKProductVer(productVersion, out buffSize);

if (result == 1)
   {
       string ProductVersion = ascii.GetString(productVersion, 0,
       buffSize - 1);
   }
```

Printer Driver Handle Functions

ZBRGetHandle

Description: Gets a handle for a printer driver.

Syntax:
```
int ZBRGetHandle(
                        HANDLE        *hPrinter,
                        char          *printerName,
                        int           *printerType,
                        int           *err)
```

Parameters:

hPrinter	[out]pointer to returned printer driver handle
printerName	[in]pointer to printer driver name
printerType	[out]pointer to returned printer type value, see Appendix B
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
IntPtr hPrinter = IntPtr.Zero;
Byte[] printerName = null;
int printerType = 0;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRPrinter.dll", EntryPoint="ZBRGetHandle", CharSet=CharSet.Auto,
   SetLastError=true)]
public static extern int ZBRGetHandle(out IntPtr hPrinter,
                                      byte[] drvName,
                                      out int printerType,
                                      out int err);

//use the function:

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();

printerName = ascii.GetBytes("Printer Driver Name");

result = ZBRGetHandle(out hPrinter, printerName, out printerType, out err);
```

ZBRCloseHandle

Description: Closes a handle for a printer driver.

Syntax:
```
int ZBRCloseHandle(
                        HANDLE        hPrinter,
                        int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRPrinter.dll", EntryPoint="ZBRCloseHandle",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRCloseHandle(IntPtr hPrinter,
                                        out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle

result = ZBRCloseHandle(hPrinter, out err);
```

Printer Driver Functions

ZBRPRNGetMsgSuppressionState

Description: Queries the printer driver for current state of status message suppression.

Syntax:
```
int ZBRPRNGetMsgSuppressionState(
                        IntPtr        Handle,
                        int           PrinterType,
                        out int       state,
                        out int       err)
```

Parameters:

Handle	[in]handle to printer driver
PrinterType	[in]printer type value, see Appendix B
state	[out]state of status message suppression: -1 = unknown 0 = suppression disabled 1 = suppression enabled
err	[out]error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Note: Call this function before sending the first job to the printer to determine the state of status message suppression if a specific suppression state is required.

Sample Code:
```
int state = 0;
int err = 0;

//import the function from the dll:
[DllImport("ZBRPrinter.dll", EntryPoint="ZBRPRNGetMsgSuppressionState",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNGetMsgSuppressionState(IntPtr hPrinter,
                                                      int PrinterType,
                                                      out int state,
                                                      out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

int alarm = ZBRPRNGetMsgSuppressionState(hPrinter, PrinterType,out state,
                                         out err);
if( alarm == 1 && err == 0) //function succeeded
{
   if(state == -1)      //suppression state unknown
   {
      //status messages may be displayed
   }
   else if (state == 0) //suppression state disabled
   {
      //status messages will be displayed
   }
   else if(state == 1)  //suppression state enabled
   {
      //status messages will not be displayed
   }
}
```

ZBRPRNSuppressStatusMsgs

Description: Sets the status message suppression state for the printer driver.

```
Syntax:      int ZBRPRNSuppressStatusMsgs(
                                IntPtr        Handle,
                                int           PrinterType,
                                int           suppressMsgs,
                                out           int err)
```

Parameters:	Handle	[in]handle to printer driver
	PrinterType	[in]printer type value, see Appendix B
	suppressMsgs	[in]status message suppression: 0 = disable suppression 1 = enable suppression
	err	[out]error code
Return Value:	1	successful
	0	failed

Error Codes: Appendix A

Note: This function needs to be called only once if the state of status message suppression does not need to be changed. Once set, the printer driver will retain the setting.

Sample Code:

```
int suppressMsgs = 1; //enable suppression
int err = 0;

//import the function from the dll:
[DllImport("ZBRPrinter.dll", EntryPoint="ZBRPRNSuppressStatusMsgs",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNSuppressStatusMsgs(IntPtr hPrinter,
                                                  int PrinterType,
                                                  int suppressMsgs,
                                                  out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

int alarm = ZBRPRNSuppressStatusMsgs(hPrinter, PrinterType,  suppressMsgs,
                                     out err);

if(alarm == 1 && err == 0) //function succeeded
    {
        //suppression state enabled
    }
```

ZBRPRNSetOverlayMode

Description: Defines the type of overlay mode and the bitmap overlay image to be used for printing. The printer driver will use this information during the printing process.

Syntax:

```
int ZBRPRNSetOverlayMode(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           side,
                        OverlayType   overlay,
                        char          *filename,
                        int           *err)
```

Parameters:

hPrinter	[in]handle to printer driver returned by ZBRGetHandle
printerType	[in]defines type of printer – returned by ZBRGetHandle
side	[in]defines the side of the card to apply the overlay 0 = front, 1 = back
overlay	[in]structure defining the type of overlay See the structure definition in Appendix C for details.
filename	[in]filename and path of the bitmap overlay image
err	[out]pointer to error code

Return Value:
1 = Success
0 = Failure

Error Codes: Appendix A

Sample Code:

```
//import the function from the dll:
   [DllImport("ZBRPrinter.dll", EntryPoint = "ZBRPRNSetOverlayMode",
             CharSet = CharSet.Auto,
             CallingConvention = CallingConvention.StdCall,
             SetLastError = true)]
   public static extern int ZBRPRNSetOverlayMode (IntPtr handle,
                                  int printerType, int side,
                                  OverlayType overlay,
                                  [MarshalAs(UnmanagedType.LPSTR)]filename,
                                  out int err);
   //use the function:
   handle     //returned by ZBRGetHandle
   printeType //returned by ZBRGetHandle
   int side = 1; //back side of card will receive overlay
   OverlayType overlay = OverlayType.MAG_STRIPE;
   String filename = "MagneticStripe.bmp";
   int err = 0;

   int result = ZBRPRNSetOverlayMode(handle, printerType, side, overlayType,
                                    overlayFile, out err);
```

Printer Command Functions

ZBRPRNPrintPrnFile

Description: Prints an *.prn file.

Syntax:
```
int ZBRPRNPrintPrnFile(
                        HANDLE        hPrinter,
                        int           printerType,
                        char          *filename,
                        int           *err)
```

Parameters:

hPrinter	[in]in]printer driver handle
printerType	[in]printer type value, see Appendix B
filename	[in]pointer to *.prn filename
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
byte[] filename = null;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNPrintPRNFile",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNPrintPRNFile(IntPtr hPrinter,
                                            int printerType,
                                            byte[] filename,
                                            out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();

filename = ascii.GetBytes("c:\\Test.prn");

result = ZBRPRNPrintPRNFile(hPrinter, printerType, filename, out err);
```

Status Functions

ZBRPRNCheckForErrors

Description: Queries the printer for its current status.

Syntax:
```
int ZBRPRNCheckForErrors(
                        IntPtr        Handle,
                        int           PrinterType)
```

Parameters:

Handle	[in]handle to printer driver
PrinterType	[in]printer type value, see Appendix B

Return Value: 0 = Printer is not in error state
>0 = Error code

Error Codes: Appendix A

Sample Code:

```
//import the function from the dll:
[DllImport("ZBRPrinter.dll", EntryPoint="ZBRPRNCheckForErrors",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNCheckForErrors(IntPtr hPrinter,
                                              int PrinterType);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

int alarm = ZBRPRNCheckForErrors(hPrinter, PrinterType);
```

Note: Call this function prior to sending a job to the printer to determine if printer is able to perform a job.

ZBRPRNClrErrStatusLn

Description: Clears the error status line.

Syntax:
```
int ZBRPRNClrErrStatusLn(
                          HANDLE        hPrinter,
                          int           printerType,
                          int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNClrErrStatusLn",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNClrErrStatusLn(IntPtr hPrinter,
                                              int printerType,
                                              out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNClrErrStatusLn(hPrinter, printerType, out err);
```

ZBRPRNGetPrintCount

Description: Gets the total number of cards printed.

Syntax:
```
int ZBRPRNGetPrintCount(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           *printCount,
                        int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
printCount	[out]	pointer to total card count
err	[out]	pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNGetPrintCount",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNGetPrintCount(IntPtr hPrinter,
                                             int printerType,
                                             out int printCount,
                                             out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNGetPrintCount(hPrinter, printerType, out printCount, out err);
```

ZBRPRNGetPrintStatus

Description: Queries the printer driver for current job status.

Syntax:
```
int ZBRPRNGetPrintStatus (
                          IntPtr        Handle,
                          int           PrinterType)
```

Parameters:

Handle	[in]handle to printer driver
PrinterType	[in]printer type value, see Appendix B

Returns:

0	= Current job is proceeding without error
>0	= Error code (See SDK Manual Appendix A)

Sample Code:

```
//import the function from the dll:
[DllImport("ZBRPrinter.dll", EntryPoint="ZBRPRNGetPrintStatus",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNGetPrintStatus(IntPtr hPrinter,
                                               int PrinterType);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

int alarm = ZBRPRNGetPrintStatus(hPrinter, PrinterType);
```

Note: Call this function after sending a job/while job is active to the printer to determine if job is being performed or has encountered an error.

ZBRPRNGetPanelsRemaining

Description: Returns the number of ribbon panels remaining.

Syntax:
```
int ZBRPRNGetPanelsRemaining(
                            HANDLE        hPrinter,
                            int           printerType,
                            int           *panels,
                            int           *err)
```

Parameters:

hPrinter	[in]printer driver handle	
printerType	[in]printer type value, see Appendix B	
panels	[out]number of panels available for printing	
err	[out]pointer to returned error code	

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int panels = 0;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNGetPanelsRemaining",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNGetPanelsRemaining(IntPtr hPrinter,
                                                  int printerType,
                                                  out int panels,
                                                  out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNGetPanelsRemaining(hPrinter, printerType, out panels, out err);
```

ZBRPRNGetPanelsPrinted

Description: Returns the number of ribbon panels that have been printed.

Syntax:
```
int ZBRPRNGetPanelsPrinted(
                            HANDLE        hPrinter,
                            int           printerType,
                            int           *panels,
                            int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
panels	[out]number of panels that have been printed
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int panels = 0;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNGetPanelsPrinted",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPNGetPanelsPrinted(IntPtr hPrinter,
                                               int printerType,
                                               out int panels,
                                               out int err);

//use the function:
//note:    hPrinter = handle returned from ZBRGetHandle
//         printerType = integer returned from ZBRGetHandle

result = ZBRPRNGetPanelsPrinted(hPrinter, printerType, out panels, out err);
```

ZBRPRNGetPrinterSerialNumber

Description: Retrieves the printer serial number.

Syntax:
```
int ZBRPRNGetPrinterSerialNumber(
                            HANDLE        hPrinter,
                            int           printerType,
                            char          *serialNum,
                            int           *respSize,
                            int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
serialNum	[out]pointer to serial number buffer
respSize	[out]pointer to buffer size
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
byte[] serialNum = new byte[50];
int respSize = 0;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNGetPrinterSerialNumber",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNGetPrinterSerialNumber(IntPtr hPrinter,
                                                      int printerType,
                                                      byte[] serialNum,
                                                      out int respSize,
                                                      out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();

result = ZBRPRNGetPrinterSerialNumber(hPrinter, printerType, serialNum,
   out respSize, out err);

if( (result == 1) && (err == 0) )
   {
       string SerialNum = ascii.GetString(serialNum, 0, respSize - 1);
   }
```

ZBRPRNGetPrinterSerialNumb

Description: Retrieves the printer serial number.

Syntax:
```
int ZBRPRNGetPrinterSerialNumb(
                            HANDLEhPrinter,
                            int           printerType,
                            char          *serialNum,
                            int           *RespSize,
                            int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
serialNum	[out]pointer to serial number buffer
RespSize	[out]pointer to buffer size
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
byte[] serialNum = new byte[50];
int respSize = 0;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNGetPrinterSerialNumb",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNGetPrinterSerialNumb(IntPtr hPrinter,
                                                    int printerType,
                                                    byte[] serialNum,
                                                    out int respSize,
                                                    out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();

result = ZBRPRNGetPrinterSerialNumb(hPrinter, printerType, serialNum,
   out respSize, out err);

if( (result == 1) && (err == 0) )
   {
       string SerialNum = ascii.GetString(serialNum, 0, respSize - 1);
   }
```

ZBRPRNGetPrinterOptions

Description: Retrieves the printer options.

Syntax:
```
int ZBRPRNGetPrinterOptions(
                        HANDLE        hPrinter,
                        int           printerType,
                        char          *options,
                        int           *respSize,
                        int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
options	[out]pointer to options buffer:

```
Base Unit Designation:
  1 = Single-sided printing
  2 - Dual-sided printing

SmartCard option
  0 = No SmartCard (SC) encoder
  A = Contact & Contactless SC encoders
  E = Contact station SC encoder

Magnetic encoder option
  0 - No magnetic encoder
  M = Magnetic encoder

Security option
  0 - No enclosure locks
  A - Enclosure locks

Interface
  0 - USB only
  C - USB & Ethernet

Firmware
  V = firmware version
```

respSize	[out]pointer to response size
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code: See next page

Sample Code:
```
byte[] options = new byte[50];
int respSize = 0;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNGetPrinterOptions",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNGetPrinterOptions(IntPtr hPrinter,
                                                 int printerType,
                                                 byte[] options,
                                                 out int respSize,
                                                 out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();

result = ZBRPRNGetPrinterOptions(hPrinter, printerType, options,
   out respSize, out err);

if( (result == 1) && (err == 0) )
   {
      string Options = ascii.GetString(options, 0, respSize - 1);
   }
```

Example Response:
```
Options = "ZXP 32AM00 V00.00.14"
          Printer configuration:
              ZXP 3 = ZXP 3 Series
                  2 = Dual-sided printing
                  A = Contact & Contactless SmartCard encoders
                  M = Magnetic encoder
                  0 = No enclosure locks
                  0 = USB only
                  V00.00.14 = firmware version 00.00.14
```

ZBRPRNGetPrintHeadSerialNumber

Description: Retrieves the print head serial number.

Syntax:
```
int ZBRPRNGetPrintHeadSerialNumber(
                                HANDLE        hPrinter,
                                int           printerType,
                                char          *serialNum,
                                int           *respSize,
                                int           *err)
```

Parameters:		
	hPrinter	[in]printer driver handle
	printerType	[in]printer type value, see Appendix B
	serialNum	[out]pointer to serial number buffer
	respSize	[out]pointer to response size
	err	[out]pointer to returned error code

Return Value:		
	1	successful
	0	failed

Error Codes: Appendix A

Sample Code:
```
byte[] serialNum = new byte[50];
int respSize = 0;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNGetPrintHeadSerialNumber",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNGetPrintHeadSerialNumber(IntPtr hPrinter,
                                                        int printerType,
                                                        byte[] serialNum,
                                                        out int respSize,
                                                        out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();

result = ZBRPRNGetPrintHeadSerialNumber(hPrinter, printerType, serialNum,
   out respSize, out err);

if( (result == 1) && (err == 0) )
   {
      string SerialNum = ascii.GetString(serialNum, 0, respSize - 1);
   }
```

ZBRPRNGetPrintHeadSerialNumb

Description: Gets the print head serial number.

Syntax:
```
int ZBRPRNGetPrintHeadSerialNumb(
                            HANDLE        hPrinter,
                            int           printerType,
                            char          *serialNum,
                            int           *respSize,
                            int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
serialNum	[out]pointer to serial number buffer
respSize,	[out]pointer to buffer size
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
byte[] serialnum = new byte[50];
int respSize = 0;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNGetPrintHeadSerialNumb",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNGetPrintHeadSerialNumb(IntPtr hPrinter,
                                                      int printerType,
                                                      byte[] serialNum,
                                                      out int respSize,
                                                      out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();

result = ZBRPRNGetPrintHeadSerialNumb(hPrinter, printerType, serialnum,
   out respSize, out err);

if( (result == 1) && (err == 0) )
   {
       string SerialNum = ascii.GetString(serialnum, 0, respSize - 1);
   }
```

ZBRPRNGetOpParam

Description: Retrieves the operational parameters.

Syntax:
```
int ZBRPRNGetOpParam(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           paramIDx,
                        char          *opParam,
                        int           *respSize,
                        int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
paramIDx	[in]requested parameter (see *Operational Parameters* below)
opParam	[out]pointer to operational parameter buffer
respSize	[out]pointer to response size
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Operational Parameters:

- 0 = Black printing parameter
- 1 = X offset
- 2 = Y offset
- 3 = Black contrast
- 4 = Varnish contrast
- 5 = Hologram contrast
- 6 = Yellow contrast
- 7 = Magenta contrast
- 8 = Cyan contrast
- 9 = K_{dye} contrast
- 10 = Yellow intensity
- 11 = Magenta intensity
- 12 = Cyan intensity
- 13 = K_{dye} intensity
- 14 = P_1 Setting for SXY Command
  - 0 = Origin offset
  - 1 = No origin offset
- 15 = Print head resistance
- 16 = Black speed
- 17 = Varnish speed
- 18 = P_1 setting for +EC
- 19 = SmartCard offset
- 20 = Magnetic encoder
  - 0 = Not connected
  - 1 = Connected
- 21 = Coercivity setting
  - 0 = LoCo
  - 1 = HiCo
- 22 = Magnetic encoding format
  - 0 = JIS2
  - 1 = ISO
- 23 = Encoder head placement
  - 0 = Below card path
  - 1 = Above card path

Sample Code: See next page

Sample Code:
```
byte[] opParam = new byte[50];
int paramIDx = 20; //determine if printer has a magnetic encoder
installed
int respSize = 0;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNGetOpParam",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNGetOpParam(IntPtr hPrinter,
                                          int printerType,
                                          int paramIDx,
                                          byte[] opParam,
                                          out int respSize,
                                          out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();

result = ZBRPRNGetOpParam(hPrinter, printerType, paramIDx, opParam,
   out respSize, out err);

if( (result == 1) && (err == 0) )
   {
      string option = ascii.GetString(opParam, 0, respSize - 1);
      if (option == "0") //no magnetic encoder installed
      if (option == "1") //magnetic encoder installed
   }
```

ZBRPRNGetPrinterStatus

Description: Returns the printer's current status code.

Note: This function only supports USB communication.

Syntax: int ZBRPRNGetPrinterStatus(
int *statusCode)

Parameters: statusCode [out]pointer to current error code status

Return Value: 1 successful
0 failed

Error Codes: Appendix A

Sample Code:

```
int statusCode = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNGetPrinterStatus",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNGetPrinterStatus(out int statusCode);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNGetPrinterStatus(out statusCode);

if(result == 1)
    {
        if(statusCode == 0)
        //printer is not in an error condition

        elseif (statusCode > 0)
        {
            //printer is in error condition
            //statusCode contains the error code for the condition
        }
    }
    else //function call failed
```

ZBRPRNGetSensorStatus

Description: Retrieves the state of various sensors for the printer.

Syntax:
```
int ZBRPRNGetSensorStatus(
                          HANDLE        hPrinter,
                          int           printerType,
                          byte          *status,
                          int           *err)
```

Parameters:

hPrinter	[in]printer driver handle	
printerType	[in]printer type value, see Appendix B	
status	[out]pointer to sensor status byte	
	bit 0 - Main Door:	
		0 = closed
		1 = open
	bit 1 - Ribbon Sensor:	
		0 = transparent/no panel
		1 = opaque panel
	bit 2 - ATM Sensor:	
		0 = clear
		1 = blocked
	bit 3 - Print Synchro Sensor:	
		0 = clear
		1 = blocked
	bit 4 - Feeder Door:	
		0 = closed
		1 = open
	bit 5 - Flipper Position:	
		0 = card feed position
		1 = card print position
	bit 6 - Card In Flipper:	
		0 = no card
		1 = card
	bit 7 - Mag Synchro Sensor:	
		0 = clear
		1 = blocked
err	[out]pointer to returned error code	

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
byte status = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNGetSensorStatus",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNGetSensorStatus(IntPtr hPrinter,
                                               int printerType,
                                               out byte status,
                                               out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNGetSensorStatus(hPrinter, printerType, out status, out err);

if( (result == 1) && (err == 0) )
   {
      //status contains the information for the sensors
   }
```

ZBRPRNIsPrinterReady

Description: Queries the print driver to determine if the printer is currently executing a print job.

Syntax:
```
int ZBRPRNIsPrinterReady(
                          HANDLE        hPrinter,
                          int           printerType,
                          int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
err	[out]pointer to returned error code

Return Value:

1	Printer is ready
0	Printer is currently executing a print job

Error Codes: Appendix A

Sample Code:
```
byte status = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNIsPrinterReady",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNIsPrinterReady(IntPtr hPrinter,
                                               int printerType,
                                               out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNIsPrinterReady(hPrinter, printerType, out err);

if( (result == 1) && (err == 0) )
   {
      //printer is ready
   }
```

Cleaning Functions

ZBRPRNChkDueForCleaning

Description: Reports current values for the Printing, Cleaning, and Cleaning Pass counters.

Syntax:
```
int ZBRPRNChkDueForClearning(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           *imgCounter,
                        int           *cleanCounter,
                        int           *cleanCardCounter,
                        int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
imgCounter	[out]pointer to total number of head-down image passes made by printer since new (note that each ribbon panel used counts as a pass)
cleanCounter	[out]pointer to current setting for image passes that trigger a cleaning alert
cleanCardCounter	[out]pointer to current setting for passes performed using a Cleaning Card
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int imgCounter = 0;
int cleanCounter = 0;
int cleanCardCounter = 0;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNChkDueForCleaning",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNChkDueForCleaning(IntPtr hPrinter,
                                                 int printerType,
                                                 out int imgCounter,
                                                 out int cleanCounter,
                                                 out int cleanCardCounter,
                                                 out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNChkDueForCleaning(hPrinter, printerType, out imgCounter,
   out cleanCounter, out cleanCardCounter, out err);
```

ZBRPRNStartCleaningSeq

Description: Starts a cleaning sequence.

Syntax:
```
int ZBRPRNStartCleaningSeq(
                            HANDLE        hPrinter,
                            int           printerType,
                            int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNStartCleaningSeq",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNStartCleaningSeq(IntPtr hPrinter,
                                                 int printerType,
                                                 out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNStartCleaningSeq(hPrinter, printerType, out err);
```

ZBRPRNStartCleaningCardSeq

Description: Starts card cleaning sequence.

Syntax:
```
int ZBRPRNSetStartCleaningCardSeq(
                            HANDLE        hPrinter,
                            int           printerType,
                            int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNStartCleaningCardSeq",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNStartCleaningCardSeq(IntPtr hPrinter,
                                                    int printerType,
                                                    out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNStartCleaningCardSeq(hPrinter, printerType, out err);
```

ZBRPRNGetCleaningParam

Description: Gets cleaning values.

Syntax:

```
int ZBRPRNGetCleaningParam(
                            HANDLE        hPrinter,
                            int           printerType,
                            int           *imgCounter,
                            int           *cleanCounter,
                            int           *cleanCardCounter,
                            int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
imgCounter	[out]pointer to total number of head-down Image passes(each ribbon panel used counts as a pass)
cleanCounter	[out]pointer to image passes before a cleaning alert is sent, default = 5000
cleanCardCounter	[out]pointer to the number of cleaning card passes when cleaning, default = 5
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:

```
int imgCounter = 0;
int cleanCounter = 0;
int cleanCardCounter = 0;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNGetCleaningParam",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNGetCleaningParam(IntPtr hPrinter,
                                                int printerType,
                                                out int imgCounter,
                                                out int cleanCounter,
                                                out int cleanCardCounter,
                                                out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNGetCleaningParam(hPrinter, printerType, out imgCounter,
   out cleanCounter, out cleanCardCounter, out err);
```

ZBRPRNSetCleaningParam

Description: Sets the cleaning parameters.

Syntax:
```
int ZBRPRNSetCleaningParam(
                            HANDLE      hPrinter,
                            int         printerType,
                            int         ribbonPanelCounter,
                            int         cleanCardPass,
                            int         *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
ribbonPanelCounter	[in]	number of panels printed before start cleaning, default = 5000
cleanCardPass	[in]	number of cleaning passes through printer, default = 5
err	[out]	pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int ribbonPanelCounter = 5000;
int cleanCardPass = 5;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNSetCleaningParam",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNSetCleaningParam(IntPtr hPrinter,
                                                int printerType,
                                                int ribbonPanelCounter,
                                                int cleanCardPass,
                                                out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNSetCleaningParam(hPrinter, printerType, ribbonPanelCounter,
   cleanCardPass, out err);
```

Printer Setup Functions

ZBRPRNResetPrinter

Description: Resets printer.

Syntax:
```
int ZBRPRNResetPrinter(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNResetPrinter",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNResetPrinter(IntPtr hPrinter,
                                            int printerType,
                                            out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNResetPrinter(hPrinter, printerType, out err);
```

ZBRPRNSelfAdj

Description: Initiates a printer self-adjust sequence.

Syntax:
```
int ZBRPRNSelfAdj(
                          HANDLE        hPrinter,
                          int           printerType,
                          int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNSelfAdj",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNSelfAdj(IntPtr hPrinter,
                                       int printerType,
                                       out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNSelfAdj(hPrinter, printerType, out err);
```

ZBRPRNGetChecksum

Description: Gets the firmware checksum.

Syntax:
```
int ZBRPRNGetChecksum(
                          HANDLE        hPrinter,
                          int           printerType,
                          int           *checksum,
                          int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
checksum	[out]	pointer to returned checksum
err	[out]	pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int checksum = 0;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNGetChecksum",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNGetChecksum(IntPtr hPrinter,
                                           int printerType,
                                           out int checksum,
                                           out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNGetChecksum(hPrinter, printerType, out checksum, out err);
```

ZBRPRNSetCardFeedingMode

Description: Sets the card feeding mode.

Syntax:
```
int ZBRPRNSetCardFeedingMode(
                            HANDLE        hPrinter,
                            int           printerType,
                            int           mode,
                            int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
mode	[in]	mode: 0 = printer with card feeder (default) 1 = printer without a card feeder
err	[out]	pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int mode = 0; //configure for printer with card feeder
int err = 0;
int result = 0

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNSetCardFeedingMode",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNSetCardFeedingMode(IntPtr hPrinter,
                                                  int printerType,
                                                  int mode,
                                                  out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNSetCardFeedingMode(hPrinter, printerType, mode, out err);
```

ZBRPRNSetPrintHeadResistance

Description: Set a printer's print head resistance.

Syntax:
```
int ZBRPRNSetPrintHeadResistance(
                            HANDLE        hPrinter,
                            int           printerType,
                            int           resistance,
                            int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
resistance	[in]	print head resistance value in ohms
err	[out]	pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Note: This function would be used following replacement of a printer's printhead. The initial value to be used is the printhead resistance displayed on the printhead's label. If adjustments to the initial value are required, do not exceed ± 200 from the label's value.

Sample Code:
```
int resistance = X; //set to value displayed on printhead label
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNSetPrintHeadResistance",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNSetPrintHeadResistance(IntPtr hPrinter,
                                                      int printerType,
                                                      int resistance,
                                                      out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNSetPrintHeadResistance(hPrinter, printerType, resistance,
    out err);
```

ZBRPRNClrMediaPath

Description: Clears a printer's media path.

Syntax:
```
int ZBRPRNClrMediaPath(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNClrMediaPath",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNClrMediaPath(IntPtr hPrinter,
                                            int printerType,
                                            out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNClrMediaPath(hPrinter, printerType, out err);
```

ZBRPRNImmediateParamSave

Description: Perfoms an immediate save of parameters to flash memory.

Syntax:
```
int ZBRPRNImmediateParamSave(
                            HANDLE        hPrinter,
                            int           printerType,
                            int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNImmediateParamSave",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNImmediateParamSave(IntPtr hPrinter,
                                                  int printerType,
                                                  out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNImmediateParamSave(hPrinter, printerType, out err
```

ZBRPRNSetRelativeXOffset

Description: Sets the X-axis Print Origin plus or minus the offset value from the current setting. Note: offset value is in dots.

Syntax:
```
int ZBRPRNSetRelativeXOffset(
                        HANDLE      hPrinter,
                        int         printerType,
                        int         offset,
                        int         *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
offset	[in]	offset value
err	[out]	pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int offset = 10;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNSetRelativeXOffset",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNSetRelativeXOffset(IntPtr hPrinter,
                                                  int printerType,
                                                  int offset,
                                                  out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNSetRelativeXOffset(hPrinter, printerType, offset, out err);
```

ZBRPRNSetRelativeYOffset

Description: Sets the Y-axis Print Origin plus or minus the offset value from the current setting. Note: offset value is in dots.

Syntax:
```
int ZBRPRNSetRelativeYOffset(
                        HANDLE      hPrinter,
                        int         printerType,
                        int         offset,
                        int         *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
offset	[in]	offset value
err	[out]	pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int offset = 10;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNSetRelativeYOffset",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNSetRelativeYOffset(IntPtr hPrinter,
                                                  int printerType,
                                                  int offset,
                                                  out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNSetRelativeYOffset(hPrinter, printerType, offset, out err);
```

ZBRPRNSetStartPrintXOffset

Description: Sets the horizontal (X-axis) start print offset point.

Syntax:
```
int ZBRPRNSetStartPrintXOffset(
                            HANDLE        hPrinter,
                            int           printerType,
                            int           offset,
                            int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
offset	[in]	offset value in dots
err	[out]	pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int offset = 10;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNSetStartPrintXOffset",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNSetStartPrintXOffset(IntPtr hPrinter,
                                                    int printerType,
                                                    int offset,
                                                    out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNSetStartPrintXOffset(hPrinter, printerType, offset, out err);
```

ZBRPRNSetStartPrintYOffset

Description: Sets the horizontal (Y-axis) start print offset point.

Syntax:
```
int ZBRPRNSetStartPrintYOffset(
                            HANDLE        hPrinter,
                            int           printerType,
                            int           offset,
                            int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
offset	[in]	offset value in dots
err	[out]	pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int offset = 10;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNSetStartPrintYOffset",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNSetStartPrintYOffset(IntPtr hPrinter,
                                                    int printerType,
                                                    int offset,
                                                    out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNSetStartPrintYOffset(hPrinter, printerType, offset, out err);
```

ZBRPRNSetStartPrintSideBOffset

Description: Alters the Horizontal (X-axis) or Vertical (Y-axis) Start Print Offset Point in dots. Note: This function is supported by dual-sided printers only.

Syntax:

```
int ZBRPRNSetStartPrintSideBOffset(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           x_y,
                        int           offset,
                        int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
x_y	[in]	X axis or Y axis 0 = X axis, 1 = Y axis
offset	[in]	offset value
err	[out]	pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:

```
int x_y = 0; //alter X-axis
int offset = 10;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNSetStartPrintSideBOffset",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNSetStartPrintSideBOffset(IntPtr hPrinter,
                                                         int printerType,
                                                         int x_y,
                                                         int offset,
                                                         out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNSetStartPrintSideBOffset(hPrinter, printerType, x_y,
   offset, out err);
```

ZBRPRNSetStartPrintSideBXOffset

Description: Sets the offset for card side B X-axis start print point.
Note: This function is supported by dual-sided printers only.

Syntax:
```
int ZBRPRNSetStartPrintSideBXOffset(
                        HANDLE      hPrinter,
                        int         printerType,
                        int         offset,
                        int         *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
offset	[in]	offset value in dots
err	[out]	pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int offset = 10;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNSetStartPrintSideBXOffset",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNSetStartPrintSideBXOffset(IntPtr hPrinter,
                                                         int printerType,
                                                         int offset,
                                                         out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNSetStartPrintSideBXOffset(hPrinter, printerType,
   offset, out err);
```

ZBRPRNSetStartPrintSideBYOffset

Description: Sets the card side B Y-axis start print offset point. Note: This function is supported by dual-sided printers only.

Syntax:
```
int ZBRPRNSetStartPrintSideBYOffset(
                        HANDLE      hPrinter,
                        int         printerType,
                        int         offset,
                        int         *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
offset	[in]offset value in dots
err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int offset = 10;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNSetStartPrintSideBYOffset",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNSetStartPrintSideBYOffset(IntPtr hPrinter,
                                                         int printerType,
                                                         int offset,
                                                         out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNSetStartPrintSideBYOffset(hPrinter, printerType,
   offset, out err);
```

ZBRPRNGetIPAddress

Description: Retrieves the IP Address for the specified Ethernet-connected printer.

Syntax:
```
int ZBRPRNGetIPAddress(
                        string        PrinterName,
                        char*         ipAddress)
```

Parameters:

PrinterName	[in]printer name assigned by printer driver
ipAddress	[out] printer's IP Address

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
string PrinterName = "Zebra ZXP Series 3 Network Card Printer1";
byte[] ipAddress = new byte[50];
int result = 0;

//import the function from the dll:
[DllImport("ZBRPrinter.dll", EntryPoint = "ZBRPRNGetIPAddress",
    CharSet = CharSet.Auto, SetLastError = true)]
public static extern int ZBRPRNGetIPAddress(
                        [MarshalAs(UnmanagedType.LPStr)]string PrinterName,
                        byte[] ipAddress);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();

result = ZBRPRNGetIPAddress(PrinterName, ipAddress);

if (result == 1)
    {
        string IPAddress = ascii.GetString(ipAddress, 0, ipAddress.Length-1);
    }
```

Image Buffer Functions

ZBRPRNSetColorContrast

Description: Sets the color contrast level for the specified image buffer.

Syntax:
```
int ZBRPRNSetColorContrast(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           imgBufIdx,
                        int           contrast,
                        int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
imgBufIdx	[in]	image buffer index: 0 = Yellow (Y) 1 = Magenta (M) 2 = Cyan (C) 3 = Dye Sublimation Black (K dye)
contrast	[in]	contrast value (0 thru 10)
err	[out]	pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int imgBufIdx = 0; //set Yellow contrast
int contrast = 5;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNSetColorContrast",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNSetColorContrast(IntPtr hPrinter,
                                                 int printerType,
                                                 int imgBufIdx,
                                                 int contrast,
                                                 out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNSetColorContrast(hPrinter, printerType, imgBufIdx,
   contrast, out err);
```

ZBRPRNSetContrastIntensityLvl

Description: Sets the color intensity level for the specified image buffer.

Syntax:
```
int ZBRPRNSetContrastIntensityLvl(
                            HANDLE        hPrinter,
                            int           printerType,
                            int           imgBufIdx,
                            int           intensity,
                            int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
imgBufIdx	[in]image buffer index: 0 = Yellow (Y) 1 = Magenta (M) 2 = Cyan (C) 3 = Dye Sublimation Black (K dye)
intensity	[in]intensity value (0 thru 10)
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int imgBufIdx = 0; //set Yellow intensity
int intensity = 5;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNSetContrastIntensityLVl",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNSetContrastIntensityLvl(IntPtr hPrinter,
                                                       int printerType,
                                                       int imgBufIdx,
                                                       int intensity,
                                                       out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNSetContrastIntensityLvl(hPrinter, printerType, imgBufIdx,
   intensity, out err);
```

ZBRPRNSetHologramIntensity

Description: Sets the hologram intensity level.

Syntax:
```
int ZBRPRNSetHologramIntensity(
                            HANDLE        hPrinter,
                            int           printerType,
                            int           intensity,
                            int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
intensity	[in]	intensity value (0 thru 10)
err	[out]	pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int intensity = 5;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNSetHologramIntensity,
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNSetHologramIntensity(IntPtr hPrinter,
                                                    int printerType,
                                                    int intensity,
                                                    out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNSetHologramIntensity(hPrinter, printerType, intensity,
   out err);
```

ZBRPRNSetMonoContrast

Description: Sets the monochrome contrast level.

Syntax: `int ZBRPRNSetMonoContrast(`

```
                            HANDLE        hPrinter,
                            int           printerType,
                            int           contrast,
                            int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
contrast	[in]	contrast value (0 thru 10)
err	[out]	pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:

```
int contrast = 5;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNSetMonoContrast,
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNSetMonoContrast(IntPtr hPrinter,
                                               int printerType,
                                               int contrast,
                                               out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNSetMonoContrast(hPrinter, printerType, contrast, out err);
```

ZBRPRNSetMonoIntensity

Description: Sets the monochrome ribbon transfer intensity level.

Syntax:
```
int ZBRPRNSetMonoIntensity(
                            HANDLE        hPrinter,
                            int           printerType,
                            int           intensity,
                            int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
intensity	[in]color contrast, from 0 to 10
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int intensity = 5;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNSetMonoIntensity,
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNSetMonoIntensity(IntPtr hPrinter,
                                                 int printerType,
                                                 int intensity,
                                                 out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNSetMonoIntensity(hPrinter, printerType, intensity, out err);
```

ZBRPRNClrSpecifiedBmp

Description: Clears the specified color buffer.

Syntax:
```
int ZBRPRNClrSpecifiedBmp(
                          HANDLE        hPrinter,
                          int           printerType,
                          int           colorBufIdx,
                          int           *err)
```

Parameters:

hPrinter	[in]printer driver handle	
printerType	[in]printer type value, see Appendix B	
colorBufIdx	[in]buffer to clear 0 = Yellow (Y) 1 = Magenta (M) 2 = Cyan (C) 3 = Dye Sublimation Black (Kdye)	
err	[out]pointer to returned error code	

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int colorBufIdx = 0; //Clear Yellow buffer
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNClrSpecifiedBmp",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNClrSpecifiedBmp(IntPtr hPrinter,
                                               int printerType,
                                               int colorBufIdx,
                                               out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNClrSpecifiedBmp(hPrinter, printerType, colorBufIdx, out err);
```

ZBRPRNClrMonoImgBuf

Description: Clears the monochrome image buffer.

Syntax:
```
int ZBRPRNClrMonoImgBuf(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           clrVarnish,
                        int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
clrVarnish	[in]	clear varnish: 0 = clear k-resin image buffer 1 = clear varnish overlay image buffer
err	[out]	pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int clrVarnish = 0; //Clear k-resin image buffer
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNClrMonoImgBuf",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNClrMonoImgBuf(IntPtr hPrinter,
                                             int printerType,
                                             int clrVarnish,
                                             out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNClrMonoImgBuf(hPrinter, printerType, clrVarnish, out err);
```

ZBRPRNClrMonoImgBufs

Description: Clears the monochrome image buffers.

Syntax:
```
int ZBRPRNClrMonoImgBufs(
                          HANDLE        hPrinter,
                          int           printerType,
                          int           clrBuffer,
                          int           *err)
```

Parameters:

hPrinter	[in]printer driver handle	
printerType	[in]printer type value, see Appendix B	
clrBuffer	[in]clear Mono Buffer 0 = K Buffer (K) 1 = Varnish Buffer (O)	
err	[out]pointer to returned error code	

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int clrBuffer = 0; //Clear K buffer
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNClrMonoImgBufs",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNClrMonoImgBufs(IntPtr hPrinter,
                                              int printerType,
                                              int clrBuffer,
                                              out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNClrMonoImgBufs(hPrinter, printerType, clrBuffer, out err);
```

ZBRPRNClrColorImgBufs

Description: Clears all of the color image buffers.

Syntax:
```
int ZBRPRNClrColorImgBufs(
                          HANDLE        hPrinter,
                          int           printerType,
                          int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNClrColorImgBufs",
CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNClrColorImgBufs(IntPtr hPrinter,
                                               int printerType,
                                               out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNClrColorImgBufs(hPrinter, printerType, out err);
```

ZBRPRNClrColorImgBuf

Description: Clears the specified color buffer.

Syntax:
```
int ZBRPRNClrColorImgBuf(
                          HANDLE        hPrinter,
                          int           printerType,
                          int           colorBufIdx,
                          int           *err)
```

Parameters:

hPrinter	[in]printer driver handle	
printerType	[in]printer type value, see Appendix B	
colorBufIdx	[in]index to the color buffer: 0 = Yellow (Y) 1 = Magenta (M) 2 = Cyan (C) 3 = Dye Sublimation Black (K dye)	
err	[out]pointer to returned error code	

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int colorBufIdx = 0; //Clear Yellow buffer
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNClrColorImgBuf",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNClrColorImgBuf(IntPtr hPrinter,
                                              int printerType,
                                              int colorBufIdx,
                                              out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNClrColorImgBuf(hPrinter, printerType, colorBufIdx, out err);
```

ZBRPRNPrintMonoImgBuf

Description: Prints the monochrome buffer and ejects the card.

Syntax:
```
int ZBRPRNPrintMonoImgBuf(
                            HANDLE        hPrinter,
                            int           printerType,
                            int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNPrintMonoImgBuf",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNPrintMonoImgBuf(IntPtr hPrinter,
                                               int printerType,
                                               out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNPrintMonoImgBuf(hPrinter, printerType, out err);
```

ZBRPRNPrintMonoImgBufEx

Description: Prints the monochrome buffer and moves the card as directed.

Syntax:

```
int ZBRPRNPrintMonoImgBufEx(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           cardCommand,
                        int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
cardCommand	[in]	post-print card command: 0 = print and eject card 10 = print and return card to print ready 20 = for Kr or Ks ribbons – print and return card to print ready, synchronizes when appropriate 30 = print and leave card in place
err	[out]	pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:

```
Int cardCommand = 0; //print and eject card
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNPrintMonoImgBufEx",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNPrintMonoImgBufEx(IntPtr hPrinter,
                                                 int printerType,
                                                 int cardCommand,
                                                 out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNPrintMonoImgBufEx(hPrinter, printerType, cardCommand,
    out err);
```

ZBRPRNPrintColorImgBuf

Description: Print the specified color image buffer.

Syntax:
```
int ZBRPRNPrintColorImgBuf(
                            HANDLE        hPrinter,
                            int           printerType,
                            int           imgBufIdx,
                            int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
imgBufIdx	[in]color image buffer index: 0 = Yellow (Y) 1 = Magenta (M) 2 = Cyan (C) 3 = Dye Sublimation Black (K dye)
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int imgBufIdx = 0; //print Yellow image buffer
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNPrintColorImgBuf",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNPrintColorImgBuf(IntPtr hPrinter,
                                                 int printerType,
                                                 int imgBufIdx,
                                                 out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNPrintColorImgBuf(hPrinter, printerType, imgBufIdx, out err);
```

ZBRPRNPrintClearVarnish

Description: Prints with clear varnish from all Image Buffers and ejects the card.

Syntax:
```
int ZBRPRNPrintClearVarnish(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNPrintClearVarnish",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNPrintClearVarnish(IntPtr hPrinter,
                                                 int printerType,
                                                 out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNPrintClearVarnish(hPrinter, printerType, out err);
```

ZBRPRNPrintVarnish

Description: Print with clear varnish.

Syntax:
```
int ZBRPRNPrintVarnish(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNPrintVarnish",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNPrintVarnish(IntPtr hPrinter,
                                            int printerType,
                                            out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNPrintVarnish(hPrinter, printerType, out err);
```

ZBRPRNPrintVarnishEx

Description: Print with clear varnish and moves the card as directed.

Syntax:
```
int ZBRPRNPrintVarnishEx(
                          HANDLE        hPrinter,
                          int           printerType,
                          int           printCardCommand,
                          int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
printCardCommand	[in]	post-print card command value: 0 = print and eject card 1 = print using inverted image buffer and eject card 10 = print and return card to print ready 11 = print using inverted image buffer and return card to print ready 30 = print and leave card in place 31 = print using inverted image buffer and leave card in place
err	[out]	pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int printCardCommand = 0; //print and eject card
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNPrintVarnishEx",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNPrintVarnishEx(IntPtr hPrinter,
                                              int printerType,
                                              int printCardCommand,
                                              out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNPrintVarnishEx(hPrinter, printerType, printCardCommand,
   out err);
```

ZBRPRNPrintHologramOverlay

Description: Prints the image data as directed and moves the card as directed.

Syntax:

```
int ZBRPRNPrintHologramOverlay(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           printCardCommand,
                        int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
printCardCommand	[in]	print command & post-print card command value: 0 = print 100% of the image buffer as hologram and eject the card 1 = print inverse of the image and eject the card 10 = print the card and return the card to print ready position
err	[out]	pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:

```
int printCardCommand = 0;  //print 100% of the image buffer as
                           //a hologram
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNPrintHologramOverlay",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNPrintHologramOverlay(IntPtr hPrinter,
                                                    int printerType,
                                                    int printCardCommand,
                                                    out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNPrintHologramOverlay(hPrinter, printerType, printCardCommand,
   out err);
```

ZBRPRNPrintCardPanel

Description: Prints from a selected color dye sublimation ribbon panel (Yellow, Magenta, Cyan, or Black) using data from an associated image buffer.

Syntax:

```
int ZBRPRNPrintCardPanel(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           imgBufIdx,
                        int           *err)
```

Parameters:

hPrinter	[in]printer driver handle	
printerType	[in]printer type value, see Appendix B	
imgBufIdx	[in]color image buffer number 0 = Yellow (Y) 1 = Magenta (M) 2 = Cyan (C) 3 = Dye Sublimation Black (Kdye)	
err	[out]pointer to returned error code	

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:

```
int imgBufIdx = 0; //print Yellow buffer
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNPrintCardPanel",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNPrintCardPanel(IntPtr hPrinter,
                                              int printerType,
                                              int imgBufIdx,
                                              out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNPrintCardPanel(hPrinter, printerType, imgBufIdx, out err);
```

ZBRPRNPrintMonoPanel

Description: Prints the monochrome buffer and ejects the card.

Syntax:
```
int ZBRPRNPrintMonoPanel(
                          HANDLE        hPrinter,
                          int           printerType,
                          int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNPrintMonoPanel",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNPrintMonoPanel(IntPtr hPrinter,
                                              int printerType,
                                              out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNPrintMonoPanel(hPrinter, printerType, out err);
```

ZBRPRNWriteBox

Description: Draws a transparent rectangle in the monochrome image buffer.

Syntax:
```
int ZBRPRNWriteBox(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           startX,
                        int           startY,
                        int           width,
                        int           height,
                        int           thickness,
                        int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
startX	[in]	start X position in dots (upper left corner X coordinate)
startY	[in]	start Y position in dots (upper left corner Y coordinate)
width	[in]	width of the box in dots
height	[in]	height of the box in dots
thickness	[in]	line thickness in dots
err	[out]	pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int startX = 0;
int startY = 0;
int width = 500;
int height = 250;
int thickness = 5;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNWriteBox",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNWriteBox(IntPtr hPrinter,
                                        int printerType,
                                        int startX,
                                        int startY,
                                        int width,
                                        int height,
                                        int thickness,
                                        out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNWriteBox(hPrinter, printerType, startX, startY, width,
   height, thickness, out err);
```

ZBRPRNWriteBoxEx

Description: Draws a transparent rectangle in the monochrome image buffer.

Syntax:

```
int ZBRPRNWriteBoxEx(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           startX,
                        int           startY,
                        int           width,
                        int           height,
                        int           thickness,
                        int           gMode,
                        int           isVarnish,
                        int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
startX	[in]	start X position in dots
startY	[in]	start Y position in dots
width	[in]	width of the box in dots
height	[in]	height of the box in dots
thickness	[in]	line thickness in dots
gMode	[in]	graphic mode: 0 = clear print area and load reverse bitmap image 1 = clear print area and load bitmap image 2 = merge bit map image with print area
isVarnish	[in]	1 = use varnish overlay 0 - do not use varnish overlay
err	[out]	pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code: See next page

Sample Code:

```
int startX = 0;
int startY = 0;
int width = 500;
int height = 250;
int thickness = 5;
int gMode = 0; // clear print area and load reversed bitmap image
int isVarnish = 1; //use varnish overlay
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNWriteBoxEx",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNWriteBoxEx(IntPtr hPrinter,
                                          int printerType,
                                          int startX,
                                          int startY,
                                          int width,
                                          int height,
                                          int thickness,
                                          int gMode,
                                          int isVarnish,
                                          out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNWriteBoxEx(hPrinter, printerType, startX, startY, width,
   height, thickness, gMode, isVarnish, out err);
```

ZBRPRNWriteText

Description: Draws a text string in the monochrome image buffer.

Syntax:

```
int ZBRPRNWriteText(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           startX,
                        int           startY,
                        int           rotation,
                        int           isBold,
                        int           height,
                        char          *text,
                        int           *err
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
startX	[in]start X position in dots
startY	[in]start Y position in dots
rotation	[in] rotation: 0 = origin lower left no rotation 1 = origin lower left 90 degrees 2 = origin lower left 180 degrees 3 = origin lower left 270 degrees 4 = origin center no rotation 5 = origin center 90 degrees 6 = origin center 180 degrees 7 = origin center 270 degrees
isBold	[in]1 = bold 0 = not bold
height	[in]height in dots of the text box: 104 = 28 point normal 140 = 28 point bold
text	[in]pointer to text buffer
err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code: See next page

Sample Code:

```
int startX = 0;
int startY = 0;
int rotation = 0; //origin lower left no rotation
int isBold = 1; //make text bold
int height = 104; //28 point normal
byte[] text = null;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNWriteText",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNWriteText(IntPtr hPrinter,
                                         int printerType,
                                         int startX,
                                         int startY,
                                         int rotation,
                                         int isBold,
                                         int height,
                                         byte[] text,
                                         out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

System.Text.ASCIIEncoding ascii = new ASCIIEncoding();

text = ascii.GetBytes("Text to write into the buffer");

result = ZBRPRNWriteText(hPrinter, printerType, startX, startY,
   rotation, isBold, height, text, out err);
```

ZBRPRNWriteTextEx

Description: Draws a text string into the monochrome image buffer.

Syntax:

```
int ZBRPRNWriteTextEx(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           startX,
                        int           startY,
                        int           rotation,
                        int           isBold,
                        int           width,
                        int           height,
                        int           gMode,
                        char          *text,
                        int           isVarnish,
                        int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
startX	[in]	start X position in dots
startY	[in]	start Y position in dots
rotation	[in]	rotation: 0 = origin lower left no rotation 1 = origin lower left 90 degrees 2 = origin lower left 180 degrees 3 = origin lower left 270 degrees 4 = origin center no rotation 5 = origin center 90 degrees 6 = origin center 180 degrees 7 = origin center 270 degrees
isBold	[in]	1 = bold 0 = not bold
width	[in]	width in dots of the text box, if 0 scales according to height
height	[in]	height in dots of the text box: 104 = 28 point normal 140 = 28 point bold
gMode	[in]	graphic mode: 0 = clear print area and load reverse bit map image 1 = clear print area and load bit map image 2 = merge bit map image with print area
text	[in]	pointer to text buffer
isVarnish	[in]	1 = use varnish overlay 0 = do not use varnish overlay
err	[out]	pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code: See next page

Sample Code:

```
int startX = 0;
int startY = 0;
int rotation = 0;  //origin lower left no rotation
int isBold = 1;    //make text bold
int width = 0;     //scale according to height
int height = 104;  //28 point normal
int gMode = 0;     //clear print area & load reverse bitmap image
byte[] text = nul1;
int isVarnish = 1; //use varnish overlay
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNWriteTextEx",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNWriteTextEx(IntPtr hPrinter,
                                           int printerType,
                                           int startX,
                                           int startY,
                                           int rotation,
                                           int isBold,
                                           int widht,
                                           int height,
                                           int gMode,
                                           byte[] text,
                                           int isVarnish, out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

System.Text.ASCIIEncoding ascii = new ASCIIEncoding();

text = ascii.GetBytes("Text to write into the buffer");

result = ZBRPRNWriteTextEx(hPrinter, printerType, startX, startY, rotation,
   isBold, width, height, gMode, text, isVarnish, out err);
```

ZBRPRNSetEndOfPrint

Description: Specifies the printing width for the x axis in dots.

Syntax:
```
int ZBRPRNSetEndOfPrint(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           xWidth
                        int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
xWidth	[in]	end of print x axis in dots
err	[out]	pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int xWidth = 300;  //x-axis width in dots
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNSetEndOfPrint",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNSetEndOfPrint(IntPtr hPrinter,
                                             int printerType,
                                             int xWidth,
                                             out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNSetEndOfPrint(hPrinter, printerType, xWidth, out err);
```

Position Card Functions

ZBRPRNMovePrintReady

Description: Moves a card to the print-ready position.

Syntax:
```
int ZBRPRNMovePrintReady(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNMovePrintReady",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNMovePrintReady(IntPtr hPrinter,
                                              int printerType,
                                              out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNMovePrintReady(hPrinter, printerType, out err);
```

ZBRPRNReversePrintReady

Description: Moves the card back to the ready position.

Syntax:
```
int ZBRPRNReversePrintReady(
                            HANDLE        hPrinter,
                            int           printerType,
                            int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNReversePrintReady",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNReversePrintReady(IntPtr hPrinter,
                                                 int printerType,
                                                 out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNReversePrintReady(hPrinter, printerType, out err);
```

ZBRPRNEjectCard

Description: Moves the card to the output hopper.

Syntax:
```
int ZBRPRNEjectCard(
                            HANDLE        hPrinter,
                            int           printerType,
                            int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNEjectCard",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNEjectCard(IntPtr hPrinter,
                                         int printerType,
                                         out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNEjectCard(hPrinter, printerType, out err);
```

ZBRPRNFlipCard

Description: Flips a card for printing, or magnetic encoding, or SmartCard encoding.

Syntax:
```
int ZBRPRNFlipCard(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
err	[out]pointer to returned error code

Note: Supported by dual-sided printers only.

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNFlipCard",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNFlipCard(IntPtr hPrinter,
                                        int printerType,
                                        out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNFlipCard(hPrinter, printerType, out err);
```

ZBRPRNMoveCard

Description: Moves the card a specified distance.

Syntax:
```
int ZBRPRNMoveCard(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           steps,
                        int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
steps	[in]	number of steps to move card (Note: 100 steps = 8mm/0.315in) a positive number moves card forward a negative number moves card backward
err	[out]	pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int steps = 100; //move the card forward 8mm
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNMoveCard",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNMoveCard(IntPtr hPrinter,
                                        int printerType,
                                        int steps,
                                        out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNMoveCard(hPrinter, printerType, steps, out err);
```

ZBRPRNMoveCardBkwd

Description: Moves the card backward by specified number of steps.

Syntax:
```
int ZBRPRNMoveCardBkwd(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           steps,
                        int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
steps	[in]	number of steps to move the card backwards (Note: 100 steps = 8mm/0.315in)
err	[out]	pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int steps = 100; //move the card backward 8mm
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNMoveCardBkwd",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNMoveCardBkwd(IntPtr hPrinter,
                                            int printerType,
                                            int steps,
                                            out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNMoveCardBkwd(hPrinter, printerType, steps, out err);
```

ZBRPRNMoveCardFwd

Description: Moves the card forward by specified number of steps.

Syntax:
```
int ZBRPRNMoveCardFwd(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           steps,
                        int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
steps	[in]number of steps to move the card forward (Note: 100 steps = 8mm/0.315in)
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int steps = 100; //move the card forward 8mm
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNMoveCardFwd",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNMoveCardFwd(IntPtr hPrinter,
                                           int printerType,
                                           int steps,
                                           out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNMoveCardFwd(hPrinter, printerType, steps, out err);
```

ZBRPRNResync

Description: Resynchronize the card under the print head.

Syntax:
```
int ZBRPRNResync(
                          HANDLE        hPrinter,
                          int           printerType,
                          int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNResync",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNResync(IntPtr hPrinter,
                                      int printerType,
                                      out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNResync(hPrinter, printerType, out err);
```

ZBRPRNStartSmartCard

Description: Positions a card for SmartCard encoding.

Syntax:
```
int ZBRPRNStartSmartCard(
                          HANDLE        hPrinter,
                          int           printerType,
                          int           cardtype,
                          int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
cardtype	[in]	type of card to be encoded: 1 = Contact 2 = Contactless 3 = UHF
err	[out]	pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int cardType = 1; //position card for contact encoding
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNStartSmartCard",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNStartSmartCard(IntPtr hPrinter,
                                               int printerType,
                                               int cardType,
                                               out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNStartSmartCard(hPrinter, printerType, cardType, out err);
```

ZBRPRNEndSmartCard

Description: Moves the SmartCard to the print ready position or ejects the card following a SmartCard encoding operation.

Syntax:
```
int ZBRPRNEndSmartCard(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           cardType,
                        int           moveType,
                        int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
cardType	[in]	SmartCard Type: 1 = Contact 2 = Contactless 3 = UHF
moveType	[in]	card movement to be performed 0 = move card to print ready position 1 = eject card
err	[out]	pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int cardType = 1; //contact SmartCard
int moveType = 0; //position card for printing
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNEndSmartCard",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNEndSmartCard(IntPtr hPrinter,
                                            int printerType,
                                            int cardType,
                                            int moveType,
                                            out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNEndSmartCard(hPrinter, printerType, cardType, moveType,
   out err);
```

Test Card Function

ZBRPRNPrintTestCard

Description: Prints a test card.

Syntax:
```
int ZBRPRNPrintTestCard(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           testcardType,
                        int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
cardType	[in]	test card type: 0 = standard test card 1 = printer test card 2 = magnetic encoder test card 3 = lamination test card
err	[out]	pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int testcardType = 0; //print standard test card
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNPrintTestCard",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNPrintTestCard(IntPtr hPrinter,
                                             int printerType,
                                             int testcardType,
                                             out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNPrintTestCard(hPrinter, printerType, testcardType, out err);
```

Barcode Card Function

ZBRPRNWriteBarCode

Description: Writes a barcode to the monochrome buffer.

Syntax:
```
int ZBRPRNWriteBarCode(
                              HANDLE        hPrinter,
                              int           printerType,
                              int           startX,
                              int           startY,
                              int           rotation,
                              int           barcodeType,
                              int           barWidthRatio,
                              int           barcodeMultiplier,
                              int           barcodeHeight,
                              int           textUnder,
                              char          *barcodeData,
                              int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
startX	[in]start X position in dots
startY	[in]start Y position in dots
rotation	[in]rotation: 0 = origin lower left and no rotation 1 = origin lower left and 90 degrees 2 = origin lower left and 180 degrees 3 = origin lower left and 270 degrees 4 = origin center and no rotation 5 = origin center and 90 degrees 6 = origin center and 180 degrees 7 = origin center and 270 degrees
barcodeType	[in]bar code type: 0 = code 39 (3 of 9 alphanumeric) 1 = 2/5 interleave (numeric, even count) 2 = 2/5 industrial (numeric, no check digit) 3 = EAN8 (numeric, 12 digits encoded) 4 = EAN13 (numeric, 12 digits encoded) 5 = UPC – A (numeric, 12 digits encoded) 6 = reserved for MONARCH 7 = code 128 C w/o check digits (numeric only, even number printed) 8 = code 128 B w/o check digits (numeric) 107 = code 128 C with check digits (numeric only, even number printed) 108 = code 128 B with check digits (numeric)
barWidthRatio	[in]bar width ratio: 0 = narrow bar = 1 dot, wide bar = 2 dots 1 = narrow bar = 1 dot, wide bar = 3 dots 2 = narrow bar = 2 dots, wide bar = 5 dots
barcodeMultiplier	[in]2 .. 9 (Appendix E)
barcodeHeight	[in]bar code height in dots (Appendix E)
textUnder	[in]text under: 1 = yes, 0 = no
barcodeData	[in]pointer to barcode buffer (Appendix E)
err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code: See next page

Sample Code:

```
int startX = 50;
int startY = 50;
int rotation = 0;          //origin lower left no rotation
int barcodeType = 0;       //code 39
int barWidthRatio = 0;     //narrow bar - 1 dot, widebar - 2 dots
int barcodeMultiplier = 2; //See appendix D for details
int barcodeHeight = 75;    //See appendix D for details
int textUnder = 0;         //do not underline text
byte[] barcodeData = null;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNWriteBarCode",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNWriteBarcode(IntPtr hPrinter,
                                            int printerType,
                                            int startX,
                                            int startY,
                                            int rotation,
                                            int barcodeType,
                                            int barWidthRatio,
                                            int barcodeMultiplier,
                                            int barcodeHeight,
                                            int textUnder,
                                            byte[] barcodeData,
                                            out int err);


//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

System.Text.ASCIIEncoding ascii = new ASCIIEncoding();

barcodeData = ascii.GetBytes("0123456789");

result = ZBRPRNWriteBarCode(hPrinter, printerType, startX, startY, rotation,
   barcodeType, barWidthRatio, barcodeMultiplier, barcodeHeight, textUnder,
   barcodeData, out err);
```

Upgrade Function

ZBRPRNUpgradeFirmware

```
Description: Downloads a firmware file to the printer.
.

Syntax:      int ZBRPRNUpgradeFirmware(
                                HANDLE        hPrinter,
                                int           printerType,
                                char          *filename,
                                int           *err)

Parameters:  hPrinter          [in ]printer driver handle
             printerType       [in ]printer type value, see Appendix B
             filename          [in ]firmware file name (binary file)
             err               [out]pointer to returned error code

Return Value:1                 successful
             0                 failed

Error Codes: Appendix A

Sample Code: string filename = "c:\\NewFirmware.bin";
             int err = 0;
             int result = 0;

//import the function from the dll:
[DllImport("ZBRPrinter.dll", EntryPoint="ZBRPRNUpgradeFirmware",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNUpgradeFirmware(IntPtr hPrinter,
                                                int printerType,
              [MarshalAs(UnmanagedType.LPStr)]string filename,
                                                out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNUpgradeFirmware(hPrinter, printerType, filename, out err);
```

Magnetic Encoder Functions

ZBRPRNSetEncodingDir

Description: Sets the magnetic encoding direction.

Syntax:
```
int ZBRPRNSetEncodingDir(
                          HANDLE        hPrinter,
                          int           printerType,
                          int           dir,
                          int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
dir	[in]	direction: 0 = forward 1 = reverse
err	[out]	pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int dir = 0; //forward
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNSetEncodingDir",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPNSetEncodingDir(IntPtr hPrinter,
                                             int printerType,
                                             int dir,
                                             out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNSetEncodingDir(hPrinter, printerType, dir, out err);
```

ZBRPRNSetTrkDensity

Description: Sets the specified track's encoding density.

Syntax:
```
int ZBRPRNSetTrkDensity(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           trkNumb,
                        int           density,
                        int           *err)
```

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
trkNumb	[in]	track number: 1 = track 1 2 = track 2 3 = track 3
density	[in]	encoding density (75 or 210)
err	[out]	pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int trkNum = 1;   //track 1
int density = 75;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNSetTrkDensity",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPNSetTrkDensity(IntPtr hPrinter,
                                            int printerType,
                                            int trkNum,
                                            int density,
                                            out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNSetTrkDensity(hPrinter, printerType, trkNum, density, out err);
```

ZBRPRNResetMagEncoder

Description: Resets the magnetic encoder.

Syntax:
```
int ZBRPRNResetMagEncoder(
                          HANDLE        hPrinter,
                          int           printerType,
                          int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNResetMagEncoder",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNResetMagEncoder(IntPtr hPrinter,
                                               int printerType,
                                               out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNResetMagEncoder(hPrinter, printerType, out err);
```

ZBRPRNSetEncoderCoercivity

Description: Sets the encoder coercivity.

Syntax:

```
int ZBRPRNSetEncoderCoercivity(
                            HANDLE        hPrinter,
                            int           printerType,
                            int           coercivity,
                            int           *err)
```

Parameters:

hPrinter	[in]printer driver handle	
printerType	[in]printer type value, see Appendix B	
coercivity	[in]coercivity: 0 = low 1 = high	
err	[out]pointer to returned error code	

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:

```
int coercivity = 0; //set to low
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNSetEncoderCoercivity",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNSetEncoderCoercivity(IntPtr hPrinter,
                                                    int printerType,
                                                    int coercivity,
                                                    out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNSetEncoderCoercivity(hPrinter, printerType, coercivity,
   out err);
```

ZBRPRNSetMagEncodingStd

Description: Sets the magnetic encoding standard.

Syntax:

```
int ZBRPRNSetMagEncodingStd(
                            HANDLE        hPrinter,
                            int           printerType,
                            int           std,
                            int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
std	[in]encoding standard: 0 = JIS 1 = ISO (default)
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:

```
int standard = 0; //set to JIS
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNSetMagEncodingStd",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNSetMagEncodingStd(IntPtr hPrinter,
                                                 int printerType,
                                                 int standard,
                                                 out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

result = ZBRPRNSetMagEncodingStd(hPrinter, printerType, standard, out err);
```

ZBRPRNEnableMagReadDataEncryption

Description: Instructs the printer to perform data masking of the magnetic data during a magnetic read operation. If enabled, the magnetic data will be encrypted before it is passed to the SDK, and the SDK will decrypt the data prior to passing it to the application. Note: once this feature is enabled, it cannot be disabled.

Syntax:

```
int ZBRPRNEnableMagReadDataEncryption(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           *err)
```

Parameters:

hPrinter	[in]handle to printer driver returned by ZBRGetHandle
printerType	[in]defines type of printer – returned by ZBRGetHandle
err	[out]pointer to error code

Return Value: 1 = Success
0 = Failure

Error Codes: Appendix A

Sample Code:

```
//import the function from the dll:
    [DllImport("ZBRPrinter.dll",
               EntryPoint = "ZBRPRNEnableMagReadDataEncryption",
               CharSet = CharSet.Auto,
               CallingConvention = CallingConvention.StdCall,
               SetLastError = true)]
    public static extern int ZBRPRNEnableMagReadDataEncryption(IntPtr handle,
                                                               int printerType,
                                                               out int err);
//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
// printerType = integer returned from ZBRGetHandle
result = ZBRPRNEnableMagReadDataEncryption(hPrinter, printerType, out err);
```

ZBRPRNReadMag

Description: Reads the specified tracks.

Syntax:

```
int ZBRPRNReadMag(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           trksToRead,
                        char          *trk1Buf,
                        int           *respSizeTrk1,
                        char          *trk2Buf,
                        int           *respSizeTrk2,
                        char          *trk3Buf,
                        int           *respSizeTrk3,
                        int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
trksToRead	[in]values ORed to determine tracks to read: 0x01 = track 1 0x02 = track 2 0x04 = track 3
trk1Buf	[out]pointer to response buffer from track 1
respSizeTrk1	[out]pointer to number of bytes returned for track 1
trk2Buf	[out]pointer to response buffer for track 2
respSizeTrk2	[out]pointer to number of bytes returned from track 2
trk3Buf	[out]pointer to response buffer for track 3
respSizeTrk3	[out]pointer to number of bytes returned from track 3
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code: See next page

Sample Code:

```
int trksToRead = 7; //read all 3 tracks
byte[] trk1Buf = new byte[50];
int respSizeTrk1 = 0;
byte[] trk2Buf = new byte[50];
int respSizeTrk2 = 0;
byte[] trk3Buf = new byte[50];
int respSizeTrk3 = 0;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNReadMag",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNReadMag(IntPtr hPrinter,
                                       int printerType,
                                       int trksToRead,
                                       byte[] trk1Buf,
                                       out int respSizeTrk1,
                                       byte[] trk2Buf,
                                       out int respSizeTrk2,
                                       byte[] trk3Buf,
                                       out int respSizeTrk3,
                                       out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();

result = ZBRPRNReadMag(hPrinter, printerType, trksToRead, trk1Buf,
   out respSizeTrk1, trk2Buf, out respSizeTrk2, trk3Buf, out respSizeTrk3,
   out err);

if( result == 1) & (err == 0) )
   {
      string track1Data = ascii.GetString(trk1Buf, 0, respSizeTrk1 - 1);
      string track2Data = ascii.GetString(trk2Buf, 0, respSizeTrk2 - 1);
      string track3Data = ascii.GetString(trk3Buf, 0, respSizeTrk3 - 1);
   }
```

ZBRPRNReadMagByTrk

Description: Reads the specified magnetic track.

Syntax:
```
int ZBRPRNReadMagByTrk(
                        HANDLE        hPrinter,
                        int           printerType,
                        int           trkNumb,
                        char          *trkBuf,
                        int           *respSize,
                        int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
trkNumb	[in]track number: 1 = track 1 2 = track 2 3 = track 3
trkBuf	[out]pointer to response buffer
respSize	[out]pointer to number of bytes returned
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int trkNum = 1; //read track 1
byte[] trkBuf = new byte[50];
int respSize = 0;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNReadMagByTrk",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNReadMagByTrk(IntPtr hPrinter,
                                            int printerType,
                                            int trkNum,
                                            byte[] trkBuf,
                                            out int respSize,
                                            out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();

result = ZBRPRNReadMagByTrk(hPrinter, printerType, trkNum, trkBuf,
   out respSize, out err);

if( result == 1) & (err == 0) )
   {
       string trackData = ascii.GetString(trkBuf, 0, respSize - 1);
   }
```

ZBRPRNWriteMag

Description: Encodes the specified tracks.

Syntax:

```
int ZBRPRNWriteMag(
                            HANDLE        hPrinter,
                            int           printerType,
                            int           trksToWrite,
                            char          *trk1Data,
                            char          *trk2Data,
                            char          *trk3Data,
                            int           *err)
```

Parameters:

hPrinter	[in]printer driver handle	
printerType	[in]printer type value, see Appendix B	
trksToWrite	[in]values ORed to determine tracks to write: 0x01 = track 1 0x02 = track 2 0x04 = track 3	
trk1Data	[in]pointer to data buffer for track 1	
trk2Data	[in]pointer to data buffer for track 2	
trk3Data	[in]pointer to data buffer for track 3	
err	[out]pointer to returned error code	

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:

```
int trksToWrite = 7; //write to all 3 tracks
byte[] trk1Data = null;
byte[] trk2Data = null;
byte[] trk3Data = null;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNWriteMag",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNWriteMag(IntPtr hPrinter,
                                        int printerType,
                                        int trksToWrite,
                                        byte[] trk1Data,
                                        byte[] trk2Data,
                                        byte[] trk3Data,
                                        out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();

string track1Data = "ABCDEFGHI";
string track2Data = "0123456789";
string track3Data = "0123456789";

trk1Data = ascii.GetBytes(track1Data);
trk2Data = ascii.GetBytes(track2Data);
trk3Data = ascii.GetBytes(track3Data);

result = ZBRPRNWriteMag(hPrinter, printerType, trksToWrite, trk1Data,
   trk2Data, trk3Data, out err);
```

ZBRPRNWriteMagByTrk

Description: Encodes data on a specified track.

Syntax:
```
int ZBRPRNWriteMagByTrk(
                          HANDLE        hPrinter,
                          int           printerType,
                          int           trkNumb,
                          char          *trkData,
                          int           *err)
```

Parameters:

hPrinter	[in]printer driver handle
printerType	[in]printer type value, see Appendix B
trkNumb	[in]track number: 1 = track 1 2 = track 2 3 = track 3
trkData	[in]pointer to data buffer
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int trkNum = 1; //write to track 1
byte[] trkData = null;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNWriteMagByTrk",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNWriteMagByTrk(IntPtr hPrinter,
                                             int printerType,
                                             int trkNum,
                                             byte[] trkData,
                                             out int err);

//use the function:
//note: hPrinter = handle returned from ZBRGetHandle
//      printerType = integer returned from ZBRGetHandle

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();

string trackData = "ABCDEFGHI";

trkData = ascii.GetBytes(trackData);

result = ZBRPRNWriteMagByTrk(hPrinter, printerType, trkNum, trkData, out err);
```

ZBRPRNWriteMagPassThru

Description: Supports the magnetic pass through commands.

Syntax:
```
int ZBRPRNWriteMagPassThru(
                          HDC           hDC,
                          int           printerType,
                          int           trkNum,
                          char          *trkData,
                          int           *err)
```

Parameters:

hDC	[in]handle to the printer's graphical context, see graphics SDK's ZBRGDIInitGraphics API for details.
printerType	[in]printer type value, see Appendix B
trkNum	[in]track number: 1 = track 1 2 = track 2 3 = track 3
trkData	[in]pointer to data buffer
err	[out]pointer to returned error code

Note: err will contain error code 40 (invalid magnetic data) if attempting to encode a track with no data.

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int trkNum = 1; //write to track 1
byte[] trkData = null;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNWriteMagPassThru",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRPRNWriteMagPassThru(IntPtr hDC,
                                                int printerType,
                                                int trkNum,
                                                byte[] trkData,
                                                out int err);

//use the function:
//note: hDC = handle returned from graphics SDK's ZBRGDIInitGraphics API
//      printerType = integer returned from ZBRGetHandle

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();

string trackData = "ABCDEFGHI";

trkData = ascii.GetBytes(trackData);

result = ZBRPRNWriteMagPassThru(hDC, printerType, trkNum, trkData, out err);
```

4

Graphic Functions

Introduction

This section contains information for software developers intending to write applications for Zebra ZXP Series 1 and ZXP Series 3 Card Printers. The application programming interface (API) provides functions to access card graphics features.

Required Skills

- Experience in developing applications for the Microsoft Windows environment
- Experience in developing applications using dynamic link libraries (dll)
- Experience with Microsoft's Windows Graphics Device Interface (GDI)

Zebra Card Printers

- ZXP Series 1
- ZXP Series 3

Communication Ports

- USB 2.0
- Ethernet

SDK Elements

- ZBRGraphics.dll
  - 32 bit dynamic link library
  - calling convention is __stdcall
- ZBRGraphics.h
- C# sample code

Installation

Directory Structure

(Disk Drive):\Zebra SDK\Graphics\#.##.## \doc
\bin
\sample

doc directory contains SDK documentation
bin directory contains the dynamic link library (dll) and include files
sample directory contains example applications

System Directories

SDK dll files should be placed in the system directory.

Example
XP
(Disk Drive):\WINDOWS\system32\

Function List

SDK Specific Function

ZBRGDIGetSDKVer

Description: Returns the SDK dll version.

Syntax:
```
void ZBRGEMGetSDKVer(
                          int         *major,
                          int         *minor,
                          int         *engLevel)
```

Parameters:

major	[out]pointer to major version number
minor	[out]pointer to minor version number
engLevel	[out]pointer to engineering level number

Return Value: None

Sample Code:
```
int major = 0;
int minor = 0;
int engLevel = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll", EntryPoint = "ZBRGDIGetSDKVer",
   CharSet = CharSet.Auto, SetLastError = true)]
public static extern void ZBRGDIGetSDKVer(out int major,
                                          out int minor,
                                          out int engLevel);

//use the function:

ZBRGDIGetSDKVer(out major, out minor, out engLevel);
```

ZBRGDIGetSDKVsn

Description: Returns the Graphics SDK version.

Syntax:
```
void ZBRGDIGetSDKVsn(
                              int          *major,
                              int          *minor,
                              int          *engLevel
```

Parameters:

major	[out]pointer to major version number
minor	[out]pointer to minor version number
engLevel	[out]pointer to engineering level number

Return Value: None

Sample Code:
```
int major = 0;
int minor = 0;
int engLevel = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll", EntryPoint = "ZBRGDIGetSDKVsn",
   CharSet = CharSet.Auto, SetLastError = true)]
public static extern void ZBRGDIGetSDKVsn(out int major,
                                          out int minor,
                                          out int engLevel);

//use the function:

ZBRGDIGetSDKVsn(out major, out minor, out engLevel);
```

ZBRGDIGetSDKProductVer

Description: Returns the SDK's product version.

Syntax:
```
int ZBRGDIGetSDKProductVer(
                            char        *productVersion,
                            int         *buffSize,
                            int         *err)
```

Parameters:

productVersion	[out]buffer to hold returned product version
buffSize	[out]size of the returned buffer in bytes
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
byte[] productVersion = new byte[50];
int buffSize = 0;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRGraphics.dll", EntryPoint="ZBRGDIGetSDKProductVer",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIGetSDKProductVer(byte[] productVersion,
                                                 out int buffSize,
                                                 out int err);

//use the function:

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();

result = ZBRGDIGetSDKProductVer(productVersion, out buffSize, out err);

if( (result == 1) && (err == 0) )
   {
      string ProductVersion = ascii.GetString(productVersion, 0,
      buffSize - 1);
   }
```

Initialization Functions

ZBRGDIInitGraphics

Description: Creates a Windows device context for a printer driver and initializes a graphic buffer for storing graphic objects.

Syntax:
```
int ZBRGDIInitGraphics(
                        char        *printerName,
                        HDC         *hDC,
                        int         *err)
```

Parameters:

printerName	[in]	printer driver assigned printer name
hDC context	[out]	pointer to returned printer device
err	[out]	pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
string DriverName = "Zebra ZXP Series 3 Network Card Printer1";
byte[] printerName = null;
IntPtr hDC = IntPtr.Zero;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIInitGraphics",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIInitGraphics(byte[] printerName,
                                            out IntPtr hDC,
                                            out int err);

//use the function:

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();
printerName = ascii.GetBytes(DriverName);

result = ZBRGDIInitGraphics(printerName, out hDC, out err);
```

ZBRGDIInitGraphicsEx

Description: Creates a Windows device context for a printer driver and initializes a graphic buffer for storing graphic objects. Starts a print job by calling StartDoc with given job name.

Syntax:

```
int ZBRGDIInitGraphicsEx(
                          char        *printerName,
                          HDC         *hDC,
                          char        *jobName,
                          int         *jobID,
                          int         *err)
```

Parameters:

printerName	[in] pointer to printer driver name
hDC	[out]pointer to returned printer device context
jobName	[in] pointer to print job name
jobID	[out]pointer to print job ID
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:

```
string DriverName = "Zebra ZXP Series 3 Network Card Printer1";
string JobName = "MyPrintJobName";
byte[] printerName = null;
byte[] jobName = null;
IntPtr hDC = IntPtr.Zero;
int jobID = 0;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIInitGraphicsEx",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIInitGraphicsEx(byte[] printerName,
                                              out IntPtr hDC,
                                              byte[] jobName,
                                              out int jobID,
                                              out int err);

//use the function:

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();
printerName = ascii.GetBytes(DriverName);
jobName = ascii.GetBytes(JobName);

result = ZBRGDIInitGraphicsEx(printerName, out hDC, jobName, out jobID,
    out err);
```

ZBRGDIInitGraphicsFromPrintDlg

Description: Creates a Windows device context from the Printer Dialog Window, initializes a graphic buffer for storing graphic objects, and calls StartDoc.

Syntax:
```
int ZBRGDIInitGraphicsFromPrintDlg(
                        HDC         *hDC,
                        int         *err)
```

Parameters:

hDC	[out]pointer to returned printer device context
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
IntPtr hDC = IntPtr.Zero;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIInitGraphicsFromPrintDlg",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIInitGraphicsFromPrintDlg(out IntPtr hDC,
                                                         out int err);

//use the function:

result = ZBRGDIInitGraphicsFromPrintDlg(out hDC, out err);
```

ZBRGDICloseGraphics

Description: Releases device context and graphic buffer memory.

Syntax:
```
int ZBRGDICloseGraphics(
                              HDC          hDC,
                              int          *err)
```

Parameters:

hDC	[in] printer device context
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDICloseGraphics",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDICloseGraphics(IntPtr hDC,
                                             out int err);

//use the function:
//note: hDC = device context returned from ZBRGDIInitGraphics,
//            ZBRGDIInitGraphicsEx, or ZBRGDIInitGraphicsFromPrintDlg

result = ZBRGDICloseGraphics(hDC, out err);
```

ZBRGDIClearGraphics

Description: Clears the graphic buffer.

Syntax: `int ZBRGDIClearGraphics(`
`int *err)`

Parameters:

err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:

```
int err = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIClearGraphics",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIClearGraphics(out int err);

//use the function:

result = ZBRGDIClearGraphics(out err);
```

ZBRGDIStartPage

Description: Calls Win32 StartPage Function.

Syntax:

```
int ZBRGDIStartPage(
                              HDC          hDC,
                              int          *err)
```

Parameters:

hDC	[in]printer device context
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:

```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIStartPage",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIStartPage(IntPtr hDC,
                                         out int err);

//use the function:
//note: hDC = device context returned from ZBRGDIInitGraphics,
//            ZBRGDIInitGraphicsEx, or ZBRGDIInitGraphicsFromPrintDlg

result = ZBRGDIStartPage(hDC, out err);
```

ZBRGDIEndPage

Description: Calls the Win32 Function EndPage.

Syntax:
```
int ZBRGDIEndPage(
                              HDC          hDC,
                              int          *err)
```

Parameters:

hDC	[in]printer device context
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIEndPage",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIEndPage(IntPtr hDC,
                                       out int err);

//use the function:
//note: hDC = device context returned from ZBRGDIInitGraphics,
//            ZBRGDIInitGraphicsEx, or ZBRGDIInitGraphicsFromPrintDlg

result = ZBRGDIEndPage(hDC, out err);
```

Print Functions

ZBRGDIPreviewGraphics

Description: Draws the graphics to a window utilizing the user supplied HWND.

Syntax:
```
int ZBRGDIPreviewGraphics(
                        HWND        hwnd,
                        int         *err)
```

Parameters:

hwnd	[in]handle to window to display image in
err	[out]pointer to returned error code

Note: ZBRGDIInitGraphics as well as the ZBRGDIDraw functions must be called prior to utilizing this function. Barcodes are not available for preview.

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
IntPtr hWnd; <= caller must supply a valid window handle for this
 parameter
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIPreviewGraphics",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIPreviewGraphics(IntPtr hWnd,
                                               out int err);

//use the function:
//note: hWnd = caller-supplied handle to window for image to be
//             displayed within

result = ZBRGDIPreviewGraphics(hWnd, out err);
```

ZBRGDIPrintGraphics

Description: Prints the contents of the graphic buffer.

Syntax:
```
int ZBRGDIPrintGraphics(
                              HDC          hDC,
                              int          *err)
```

Parameters:

hDC	[in] printer device context
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIPrintGraphics",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIPrintGraphics(IntPtr hDC,
                                             out int err);

//use the function:
//note: hDC = device context returned from ZBRGDIInitGraphics,
//            ZBRGDIInitGraphicsEx, or ZBRGDIInitGraphicsFromPrintDlg

result = ZBRGDIPrintGraphics(hDC, out err);
```

ZBRGDIPrintGraphicsEx

Description: Prints the contents of the internal graphics buffer to the given device context. The difference between **ZBRGDIPrintGraphics** and this function is that this function only draws graphics in hDC buffer without putting them between the Win32 StartPage and EndPage calls.

Syntax:

```
int ZBRGDIPrintGraphicsEx(
                        HDC         hDC,
                        int         *err)
```

Parameters:

hDC	[in]printer device context
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:

```
int err = 0;
int result = 0;

//import the function from the dll:
   [DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIPrintGraphicsEx",
CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIPrintGraphicsEx(IntPtr hDC,
                                               out int err);

//use the function:
//note: hDC = device context returned from ZBRGDIInitGraphics,
//            ZBRGDIInitGraphicsEx, or ZBRGDIInitGraphicsFromPrintDlg

result = ZBRGDIPrintGraphicsEx(hDC, out err);
```

ZBRGDIPrintFilePos

Description: Prints an image file.

Syntax:
```
int ZBRGDIPrintFilePos(
                          HDC         hDC,
                          char        *filename,
                          int         position,
                          int         *err)
```

Parameters:

hDC	[in] printer device context
filename	[in] pointer to image filename
position	[in] position: 0 = ZBR_UPPER_LEFT 1 = ZBR_LOWER_LEFT 2 = ZBR_UPPER_RIGHT 3 = ZBR_LOWER_RIGHT 4 = ZBR_CENTERED
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
string fileName = "C:\\MyImageFile.bmp";
byte[] fName = null;
int position = 0; //ZBR_UPPER_LEFT
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIPrintFilePos",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIPrintFilePos(IntPtr hDC,
                                            byte[] filename,
                                            int position,
                                            out int err);

//use the function:
//note: hDC = device context returned from ZBRGDIInitGraphics,
//            ZBRGDIInitGraphicsEx, or ZBRGDIInitGraphicsFromPrintDlg

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();
fName = ascii.GetBytes(fileName);

result = ZBRGDIPrintFilePos(hDC, fName, position, out err);
```

ZBRGDIPrintFileRect

Description: Prints an image file within the rectangle boundaries.

Syntax:
```
int ZBRGDIPrintFileRect(
                          HDC         hDC,
                          char        *filename,
                          int         x,
                          int         y,
                          int         width,
                          int         height,
                          int         *err)
```

Parameters:

hDC	[in]	printer device context
filename	[in]	pointer to image filename
x	[in]	x position of the top-left corner
y	[in]	y position of the top-left corner
width	[in]	rectangle width in dots
height	[in]	rectangle height in dots
err	[out]	pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
string fileName = "C:\\MyImageFile.bmp";
byte[] fName = null;
int x = 0;
int y = 0;
int width = 500;
int height = 250;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIPrintFileRect",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIPrintFileRect(IntPtr hDC,
                                             byte[] filename,
                                             int x, int y,
                                             int width, int height,
                                             out int err);

//use the function:
//note: hDC = device context returned from ZBRGDIInitGraphics,
//            ZBRGDIInitGraphicsEx, or ZBRGDIInitGraphicsFromPrintDlg

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();
fName = ascii.GetBytes(fileName);

result = ZBRGDIPrintFileRect(hDC, fName, x, y, width, height, out err);
```

ZBRGDIIsPrinterReady

Description: Queries the Print Queue to determine if the printer is currently executing a print job.

Syntax:

```
int ZBRGDIIsPrinterReady(
                        char        *printerName,
                        int         *err)
```

Parameters:

printerName	[in]	pointer to printer driver name
err	[out]	pointer to returned error code

Return Value:

1	Printer is ready
0	Printer is currently executing a print job

Error Codes: Appendix A

Note: If ZBRGDIInitGraphics or ZBRGDIInitGraphicsFromPrintDlg is called prior to this function, the printerName parameter may be set to “null” or ""; however, if the HDC is initialized outside of the Graphics SDK, the printerName parameter must be set to the valid print driver name.

Sample Code:

```
string DriverName = "Zebra ZXP Series 3 Network Card Printer1";
byte[] printerName = null;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIIsPrinterReady",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIIsPrinterReady(byte[] printerName,
                                              out int err);

//use the function:

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();
printerName = ascii.GetBytes(DriverName);

result = ZBRGDIIsPrinterReady(printerName, out err);
```

ZBRGDIPrintImagePos

Description: Prints the contents of the graphics buffer at the specified position, and ejects the card.

Syntax:

```
int ZBRGDIPrintImagePos(
                        HDC         hDC,
                        byte        *imageData,
                        int         imageSize,
                        int         position,
                        int         *err)
```

Parameters:

hDC	[in]	printer device context
imageData	[in]	pointer to image buffer
imageSize	[in]	size of the image buffer
position	[in]	position of image: 0 = upper left 1 = lower left 2 = upper right 3 = lower right 4 = centered
err	[out]	pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:

```
string imageFile = "c:\\MyImageFile.bmp";
byte[] imageData = null;
int imageSize = 0;
int position = 0; //upper left
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIPrintImagePos",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIPrintImagePos(IntPtr hDC,
                                             byte[] imageData,
                                             int imageSize,
                                             int position,
                                             out int err);

//use the function:
//note: hDC = device context returned from ZBRGDIInitGraphics,
//            ZBRGDIInitGraphicsEx, or ZBRGDIInitGraphicsFromPrintDlg

Image img = System.Drawing.Image.FromFile(imageFile);
MemoryStream ms = new MemoryStream();
img.Save(ms, System.Drawing.Imaging.ImageFormat.Bmp);

imageData = ms.ToArray();
imageSize = imageData.GetLength(0);

result = ZBRGDIPrintImagePos(hDC, imageData, imageSize, position, out err);
```

ZBRGDIPrintImageRect

Description: Prints the contents of the graphics buffer within the specified rectangular region, and ejects the card.

Syntax:
```
int ZBRGDIPrintImageRect(
                          HDC         hDC,
                          byte        *imageData,
                          int         imageSize,
                          int         x,
                          int         y,
                          int         width,
                          int         height,
                          int         *err)
```

Parameters:

hDC	[in] printer device context
imageData	[in] pointer to image buffer
imageSize	[in] size of the image buffer
x	[in] x coordinate of image upper-left corner
y	[in] y coordinate of image upper-left corner
width	[in] image width
height	[in] image height
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
string imageFile = "c:\\MyImageFile.bmp";
byte[] imageData = null;
int imageSize = 0;
int x = 0; //upper left corner of card
int y = 0; //upper left corner of card
int width = 500;
int height = 250;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIPrintImageRect",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIPrintImageRect(IntPtr hDC,
                                               byte[] imageData,
                                               int imageSize,
                                               int x,
                                               int y,
                                               int width,
                                               int height,
                                               out int err);
//use the function:
//note: hDC = device context returned from ZBRGDIInitGraphics,
//            ZBRGDIInitGraphicsEx, or ZBRGDIInitGraphicsFromPrintDlg

Image img = System.Drawing.Image.FromFile(imageFile);
MemoryStream ms = new MemoryStream();
img.Save(ms, System.Drawing.Imaging.ImageFormat.Bmp);

imageData = ms.ToArray();
imageSize = imageData.GetLength(0);

result = ZBRGDIPrintImageRect(hDC, imageData, imageSize, x, y, width, height,
    out err);
```

ZBRGDICancelJob

Description: Cancels a job in the spooler.

Syntax:
```
int ZBRGDICancelJob(            char *printerName
                                int jobID,
                                int *err)
```

Parameters:

printerName	[in]	printer name assigned by printer driver instance
jobID	[in]	job to cancel
err	[out]	pointer to returned error code

Return Value:
1 successful
0 failed

Error Codes: Appendix A

Sample Code:
```
int err = 0;
int result = 0;
string printerName = "ZXP Series 3 USB Card Printer";

//import the function from the dll:
[DllImport( "ZBRGraphics.dll", EntryPoint="ZBRGDICancelJob",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDICancelJob (
                                        byte[] printerName,
                                        int jobID,
                                        out int err);
//use the function:
// jobID = integer returned from ZBRGDIInitGraphicsEx
byte[] name = ASCIIEncoding.ASCII.GetBytes(printerName);

result = ZBRGDICancelJob(name, jobID, out err);
```

Draw Functions

ZBRGDIDrawText

Description: Draws text in the graphic buffer.

Syntax:

```
int ZBRGDIDrawText(
                        int         x,
                        int         y,
                        char        *text,
                        char        *font,
                        int         fontSize,
                        int         fontStyle,
                        int         color,
                        int         *err)
```

Parameters:

x	[in] x position, top-left corner of text
y	[in] y position, top-left corner of text
text	[in] pointer to text buffer
font	[in] pointer to font name
fontSize	[in] point size
fontStyle	[in] values ORed to form font style: 0x01 = bold 0x02 = italic 0x04 = underline 0x08 = strikethrough
color	[in] RGB value
err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code: See next page

Sample Code:

```
int x = 0;
int y = 0;
string TextToPrint = "Printed Text";
byte[] text = null;
string FontToUse = "Arial";
byte[] font = null;
int fontSise = 12;
int fontStyle = 1; //bold
int color = 0x0FF0000; //black
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIDrawText",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIDrawText(int x,
                                        int y,
                                        byte[] text,
                                        byte[] font,
                                        int fontSize,
                                        int fontStyle,
                                        int color,
                                        out int err);

//use the function:

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();
text = ascii.GetBytes(TextToPrint);
font = ascii.GetBytes(FontToUse);

result = ZBRGDIDrawText(x, y, text, font, fontSize, fontStyle, color,
   out err);
```

ZBRGDIDrawTextEx

Description: Draws text in the graphics buffer with specified rotation and alignment.

Syntax:

```
int ZBRGDIDrawTextEx(
                    int           x,
                    int           y,
                    int           angle,
                    int           alignment,
                    char          *text,
                    char          *font,
                    int           fontSize,
                    int           fontStyle,
                    int           color,
                    int           *err)
```

Parameters:

x	[in] x position, top-left corner of text
y	[in] y position, top-left corner of text
angle	[in] text rotation angle with (x,y) as rotation origin
alignment	[in] text alignment across (x,y)
origin	4 = center justified 5 = left justified 6 = right justified
text	[in] pointer to text buffer
font	[in] pointer to font name
fontSize	[in] point size
fontStyle	[in] values ORed to form font style: 0x01 = bold 0x02 = italic 0x04 = underline 0x08 = strikethrough
color	[in] RGB value
err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code: See next page

Sample Code:

```
int x = 0;
int y = 0;
int angle = 0;     //0 degrees rotation (no rotation)
int alignment = 4; //center justified
string TextToPrint = "Printed Text";
byte[] text = null;
string FontToUse = "Arial";
byte[] font = null;
int fontSise = 12;
int fontStyle = 1; //bold
int color = 0x0FF0000; //black
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIDrawTextEx",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIDrawTextEx(int x,
                                          int y,
                                          int angle,
                                          int alignment,
                                          byte[] text,
                                          byte[] font,
                                          int fontSize,
                                          int fontStyle,
                                          int color,
                                          out int err);

//use the function:

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();
text = ascii.GetBytes(TextToPrint);
font = ascii.GetBytes(FontToUse);

result = ZBRGDIDrawTextEx(x, y, angle, alignment, text, font, fontSize,
    fontStyle, color, out err);
```

ZBRGDIDrawTextUnicode

Description: Draws Unicode text in the graphic buffer.

Syntax:
```
int ZBRGDIDrawTextUnicode(
                          int           x,
                          int           y,
                          wchar         *text,
                          wchar         *font,
                          int           fontSize,
                          int           fontStyle,
                          int           color,
                          int           *err)
```

Parameters:

x	[in] x position, top-left corner of text
y	[in] y position, top-left corner of text
text	[in] pointer to text buffer
font	[in] pointer to font name
fontSize	[in] point size
fontStyle	[in] values ORed to form font style: 0x01 = bold 0x02 = italic 0x04 = underline 0x08 = strikethrough
color	[in] RGB value
err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int x = 0;
int y = 0;
string TextToPrint = "Printed Text";
byte[] text = null;
string FontToUse = "Arial";
byte[] font = null;
int fontSise = 12;
int fontStyle = 1; //bold
int color = 0x0FF0000; //black
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIDrawTextUnicode",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIDrawTextUnicode(int x,
                                               int y,
                                               byte[] text,
                                               byte[] font,
                                               int fontSize,
                                               int fontStyle,
                                               int color,
                                               out int err);
//use the function:

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();
text = ascii.GetBytes(TextToPrint);
font = ascii.GetBytes(FontToUse);

result = ZBRGDIDrawText(x, y, text, font, fontSize, fontStyle, color,
   out err);
```

ZBRGDIDrawTextUnicodeEx

Description: Draws Unicode text in the graphics buffer with specified rotation and alignment.

Syntax:
```
int ZBRGDIDrawTextUnicode(
                    int x,
                    int y,
                    int angle,
                    int alignment,
                    wchar *text,
                    wchar *font,
                    int fontSize,
                    int fontStyle,
                    int color,
                    int *err)
```

Parameters:

x	[in] x position, top-left corner of text
y	[in] y position, top-left corner of text
angle	[in] text rotation angle with (x,y) as rotation origin
alignment	[in] text alignment across (x,y)
origin	4 = center justified 5 = left justified 6 = right justified
text	[in] pointer to text buffer
font	[in]pointer to font name
fontSize	[in] point size
fontStyle	[in] values ORed to form font style: 0x01 = bold 0x02 = italic 0x04 = underline 0x08 = strikethrough
color	[in] RGB value
err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code: See next page

Sample Code:

```
int x = 0;
int y = 0;
int angle = 0;     //0 degrees rotation (no rotation)
int alignment = 4; //center justified
string TextToPrint = "Printed Text";
byte[] text = null;
string FontToUse = "Arial";
byte[] font = null;
int fontSise = 12;
int fontStyle = 1; //bold
int color = 0x0FF0000; //black
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIDrawTextUnicodeEx",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIDrawTextUnicodeEx(int x,
                                                 int y,
                                                 int angle,
                                                 int alignment,
                                                 byte[] text,
                                                 byte[] font,
                                                 int fontSize,
                                                 int fontStyle,
                                                 int color,
                                                 out int err);

//use the function:

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();
text = ascii.GetBytes(TextToPrint);
font = ascii.GetBytes(FontToUse);

result = ZBRGDIDrawTextUnicodeEx(x, y, angle, alignment, text, font, fontSize,
    fontStyle, color, out err);
```

ZBRGDIDrawTextRect

Description: Draws text in the graphic buffer within the rectangle boundaries.

Syntax:

```
int ZBRGDIDrawTextRect(
                        int         x,
                        int         y,
                        int         width,
                        int         height,
                        int         alignment,
                        char        *text,
                        char        *font,
                        int         fontSize,
                        int         fontStyle,
                        int         color,
                        int         *err)
```

Parameters:

x	[in] x position, top-left corner of rectangle
y	[in] y position, top-left corner of rectangle
width	[in] rectangle width in dots
height	[in] rectangle height in dots
alignment	[in] alignment: 4 = center justified 5 = left justified 6 = right justified
text	[in] pointer to text buffer
font	[in] pointer to font name
fontSize	[in] point size
fontStyle	[in] values ORed to form font style: 0x01 = bold 0x02 = italic 0x04 = underline 0x08 = strikethrough
color	[in] RGB value
err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code: See next page

Sample Code:

```
int x = 0;
int y = 0;
int width = 500;
int height = 250;
int alignment = 4; //center justified
string TextToPrint = "Printed Text";
byte[] text = null;
string FontToUse = "Arial";
byte[] font = null;
int fontSise = 12;
int fontStyle = 1;     //bold
int color = 0x0FF0000; //black
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIDrawTextRect",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIDrawTextRect(int x,
                                            int y,
                                            int width,
                                            int height,
                                            int alignment,
                                            byte[] text,
                                            byte[] font,
                                            int fontSize,
                                            int fontStyle,
                                            int color,
                                            out int err);

//use the function:

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();
text = ascii.GetBytes(TextToPrint);
font = ascii.GetBytes(FontToUse);

result = ZBRGDIDrawTextRect(x, y, width, height, alignment, text, font,
   fontSize, fontStyle, color, out err);
```

ZBRGDIDrawTextRectEx

Description: Draws the given text in the graphics buffer within the bounds of the specified rectangle with specified rotation and alignment. Note: The text will be automatically moved to the next line if it does not fit the given width.

Syntax:

```
int ZBRGDIDrawTextRectEx(
                    int         x,
                    int         y,
                    int         width,
                    int         height,
                    int         angle,
                    int         alignment,
                    char        *text,
                    char        *font,
                    int         fontSize,
                    int         fontStyle,
                    int         color,
                    int         *err)
```

Parameters:

x	[in] x coordinate of the rectangle bounding the text
y	[in] y coordinate of the rectangle bounding the text
width	[in] width of the rectangle bounding the text
height	[in] height of the rectangle bounding the text
angle	[in] text rotation angle
alignment	[in] alignment: 4 = center justified 5 = left justified 6 = right justified
text	[in] pointer to text that needs to be drawn
font	[in] pointer to font name
fontsize	[in] font size
fontStyle	[in] values ORed to form font style: 0x01 = bold 0x02 = italic 0x04 = underline 0x08 = strikethrough
color	[in] RGB value
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code: See next page

Sample Code:

```
int x = 0;
int y = 0;
int width = 500;
int height = 250;
int angle = 0;     //0 degrees rotation (no rotation)
int alignment = 4; //center justified
string TextToPrint = "Printed Text";
byte[] text = null;
string FontToUse = "Arial";
byte[] font = null;
int fontSise = 12;
int fontStyle = 1; //bold
int color = 0x0FF0000; //black
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIDrawTextRectEx",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIDrawTextRectEx(int x,
                                              int y,
                                              int width,
                                              int height,
                                              int angle,
                                              int alignment,
                                              byte[] text,
                                              byte[] font,
                                              int fontSize,
                                              int fontStyle,
                                              int color,
                                              out int err);

//use the function:

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();
text = ascii.GetBytes(TextToPrint);
font = ascii.GetBytes(FontToUse);

result = ZBRGDIDrawTextRectEx(x, y, width, height, angle, alignment, text,
    font, fontSize, fontStyle, color, out err);
```

ZBRGDIDrawLine

Description: Draws a line in the graphic buffer.

Syntax:
```
int ZBRGDIDrawLine(
                              int         x1,
                              int         y1,
                              int         x2,
                              int         y2,
                              int         color,
                              float       thickness,
                              int         *err)
```

Parameters:

x1	[in] starting x position for the line in dots
y1	[in] starting y position for the line in dots
x2	[in] ending x position for the line in dots
y2	[in] ending y position for the line in dots
color	[in] RGB value
thickness	[in] thickness in dots
err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int x1 = 0;
int y1 = 0;
int x2 = 500;
int y2 = 0;
int color = 0x0FF0000; //black
float thickness = 100.0;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIDrawLine",
CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIDrawLine(int x1,
                                        int y1,
                                        int x2,
                                        int y2,
                                        int color,
                                        float thickness,
                                        out int err);

//use the function:

result = ZBRGDIDrawLine(x1, y1, x2, y2, color, thickness, out err);
```

ZBRGDIDrawImage

Description: Places a file image in the graphic buffer.

Syntax:
```
int ZBRGDIDrawImage(
                            char        *filename,
                            int         x,
                            int         y,
                            int         *err)
```

Parameters:

filename	[in] pointer to name of the file that contains the image
x	[in] x position, top-left corner of image
y	[in] y position, top-left corner of image
err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
string FileName = "c:\\MyImageFile.bmp";
byte[] fName = null;
int x = 0;
int y = 0;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIDrawImage",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIDrawImage(byte[] filename,
                                         int x,
                                         int y,
                                         out int err);

//use the function:

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();
fName = ascii.GetBytes(FileName);

result = ZBRGDIDrawImage(fName, x, y, out err);
```

ZBRGDIDrawImageEx

Description: Places a file image in the graphics buffer.

Syntax:
```
int ZBRGDIDrawImageEx(
                            char        *filename,
                            int         imageSize,
                            int         x,
                            int         y,
                            int         *err)
```

Parameters:

filename	[in] pointer to name of the file that contains the image
imageSize	[in] size of the image
x	[in] x position, top-left corner of image
y	[in] y position, top-left corner of image
err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
string FileName = "c:\\MyImageFile.bmp";
byte[] fName = null;
int imageSize = 0;
int x = 0; //top-left corner of card
int y = 0; //top-left corner of card
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIDrawImageEx",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIDrawImageEx(byte[] filename,
                                           int imageSize,
                                           int x,
                                           int y,
                                           out int err);

//use the function:

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();
fName = ascii.GetBytes(FileName);
imageSize = fName.GetLength(0);

result = ZBRGDIDrawImageEx(fName, imageSize, x, y, out err);
```

ZBRGDIDrawImagePos

Description: Places a file image in the graphic buffer.

Syntax:

```
int ZBRGDIDrawImagePos(
                        char        *filename,
                        int         position,
                        int         *err)
```

Parameters:

	filename	[in] pointer to name of the file that contains the image
	position	[in] position 0 = upper left 1 = lower left 2 = upper right 3 = lower right 4 = centered
	err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:

```
string FileName = "c:\\MyImageFile.bmp";
byte[] fName = null;
int position = 0; //upper left
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIDrawImagePos",
CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIDrawImagePos(byte[] filename,
                                            int position,
                                            out int err);

//use the function:
System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();
fName = ascii.GetBytes(FileName);

result = ZBRGDIDrawImagePos(fName, position, out err);
```

ZBRGDIDrawImagePosEx

```
Description: Places a file image in the graphics buffer.

Syntax:      int ZBRGDIDrawImagePosEx(
                                char        *filename,
                                int         imageSize,
                                int         position,
                                int         *err)

Parameters:  filename            [in] pointer to name of the file that
             contains the image
             imageSize                      [in] size of the image
             position                       [in] position
                                            0 = upper left
                                            1 = lower left
                                            2 = upper right
                                            3 = lower right
                                            4 = centered
             err                 [out]pointer to returned error code

Return Value: 1                  successful
              0                  failed

Error Codes: Appendix A

Sample Code: string FileName = "c:\\MyImageFile.bmp";
             byte[] fName = null;
             int imageSize = 0;
             int position = 0; //upper left
             int err = 0;
             int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIDrawImagePosEx",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIDrawImagePosEx(byte[] filename,
                                               int imageSize,
                                               int position,
                                               out int err);

//use the function:

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();
fName = ascii.GetBytes(FileName);
imageSize = fName.GetLength(0);

result = ZBRGDIDrawImagePosEx(fName, imageSize, position, out err);
```

ZBRGDIDrawImageRect

Description: Places a file image in the graphic buffer within rectangle boundaries.

Syntax:
```
int ZBRGDIDrawImageRect(
                        char        *filename,
                        int         x,
                        int         y,
                        int         width,
                        int         height,
                        int         *err)
```

Parameters:

filename	[in] pointer to name of the file that contains the image
x	[in] x position, top-left corner of rectangle
y	[in] y position, top-left corner of rectangle
width	[in] rectangle width in dots
height	[in] rectangle height in dots
err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
string FileName = "c:\\MyImageFile.bmp";
byte[] fName = null;
int x = 0;
int y = 0;
int width = 500;
int height = 250;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIDrawImageRect",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIDrawImageRect(byte[] filename,
                                             int x,
                                             int y,
                                             int width,
                                             int height,
                                             out int err);

//use the function:

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();
fName = ascii.GetBytes(FileName);

result = ZBRGDIDrawImageRect(fName, x, y, width, height, out err);
```

ZBRGDIDrawImageRectEx

Description: Places a file image in the graphics buffer within the specified rectangular region.

Syntax:

```
int ZBRGDIDrawImageRectEx(
                          char        *filename,
                          int         imageSize,
                          int         x,
                          int         y,
                          int         width,
                          int         height,
                          int         *err)
```

Parameters:

filename	[in] pointer to name of the file that contains the image
imageSize	[in] size of the image
x	[in] x coordinate of image upper-left corner
y	[in] y coordinate of image upper-left corner
width	[in] image width
height	[in] image height
err	[out]pointer to returned error code

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:

```
string FileName = "c:\\MyImageFile.bmp";
byte[] fName = null;
int imageSize = 0;
int x = 0; //upper left corner
int y = 0; //upper left corner
int width = 500;
int height = 250;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIDrawImageRectEx",
    CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIDrawImageRectEx(byte[] filename,
                                               int imageSize,
                                               int x,
                                               int y,
                                               int width,
                                               int height,
                                               out int err);

//use the function:

System.Text.ASCIIEncoding ascii = new System.Text.ASCIIEncoding();
fName = ascii.GetBytes(FileName);
imageSize = fName.GetLength(0);

result = ZBRGDIDrawImageRectEx(fName, imageSize, x, y, width, height,
    out err);
```

ZBRGDIDrawRectangle

Description: Draws a rectangle in the graphic buffer.

Syntax:
```
int ZBRGDIDrawRectangle(
                        int         x,
                        int         y,
                        int         width,
                        int         height,
                        float       thickness,
                        int         color,
                        int         *err)
```

Parameters:

x	[in] x position, top-left corner of rectangle
y	[in] y position, top-left corner of rectangle
width	[in] rectangle width in dots
height	[in] rectangle height in dots
thickness	[in] line thickness for the rectangle
color	[in] RGB color value
err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int x = 0;
int y = 0;
int width = 500;
int height = 250;
float thickness = 50;
int color = 0x0FF0000; //black
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIDrawRectangle",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIDrawRectangle(int x,
                                             int y,
                                             int width,
                                             int height,
                                             float thickness,
                                             int color,
                                             out int err);

//use the function:

result = ZBRGDIDrawRectangle(x, y, width, height, thickness, color, out err);
```

ZBRGDIDrawEllipse

Description: Draws an ellipse in the graphic buffer.

Syntax:
```
int ZBRGDIDrawEllipse(
                        int         x,
                        int         y,
                        int         width,
                        int         height,
                        float       thickness,
                        int         color,
                        int         *err)
```

Parameters:

x	[in] x position, top-left corner of rectangle
y	[in] y position, top-left corner of rectangle
width	[in] width of the ellipse
heigth	[in] height of the ellipse
thickness	[in] line thickness for the ellipse in dots
color	[in] RGB color value
err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code:
```
int x = 0;
int y = 0;
int width = 500;
int height = 250;
float thickness = 50;
int color = 0x0FF0000; //black
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIDrawEllipse",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIDrawEllipse(int x,
                                           int y,
                                           int width, int height,
                                           float thickness,
                                           int color, out int err);

//use the function:

result = ZBRGDIDrawEllipse(x, y, width, height, thickness, color, out err);
```

ZBRGDIDrawBarCode

Description: Writes a barcode to the monochrome buffer.

Syntax:
```
int ZBRGDIDrawBarCode(
                    int           X,
                    int           Y,
                    int           rotation,
                    int           barcodeType,
                    int           barWidthRatio,
                    int           barcodeMultiplier,
                    int           barcodeHeight,
                    int           textUnder,
                    char          *barcodeData,
                    int           *err)
```

Parameters:

X	[in] start X position in dots
Y	[in] start Y position in dots
rotation	[in] rotation: 0 = origin lower left no rotation 1 = origin lower left 90 degrees 2 = origin lower left 180 degrees 3 = origin lower left 270 degrees 4 = origin center no rotation 5 = origin center 90 degrees 6 = origin center 180 degrees 7 = origin center 270 degrees
barcodeType	[in] bar code type: 0 = code 39 (3 of 9 alphanumeric) 1 = 2/5 interleave (numeric, even, no count) 2 = 2/5 industrial (numeric, no check digit) 3 = EAN8 (numeric 12 digits encoded) 4 = EAN13 (numeric 12 digits encoded) 5 = UPC – A (numeric 12 digits encoded) 6 = reserved for MONARCH 7 = code 128 C w/o check digits (numeric only, even number printed) 8 = code 128 B w/o check digits (numeric) 107 = code 128 C with check digits (numeric only, even number printed) 108 = code 128 B with check digits (numeric)
barWidthRatio	[in] bar width ratio: 0 = narrow bar = 1 dot, wide bar = 2 dots 1 = narrow bar = 1 dot, wide bar = 3 dots 2 = narrow bar = 2 dots, wide bar = 5 dots
barcodeMultiplier	[in] barcode multiplier
barcodeHeight	[in] bar code height in dots
textUnder	[in] text under: 1 = yes 0 = no
barcodeData	[in] pointer to barcode data bbuffer
err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

1	successful
0	failed

Error Codes: Appendix A

Sample Code: See next page

Sample Code:

```
int X = 50;
int Y = 50;
int rotation = 0;          //origin lower left no rotation
int barcodeType = 0;       //code 39
int barWidthRatio = 0;     //narrow bar - 1 dot, widebar - 2 dots
int barcodeMultiplier = 2; //See appendix D for details
int barcodeHeight = 75;    //See appendix D for details
int textUnder = 0;         //do not underline text
byte[] barcodeData = null;
int err = 0;
int result = 0;

//import the function from the dll:
[DllImport( "ZBRGraphics.dll", EntryPoint="ZBRGDIDrawBarCode",
   CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIDrawBarcode(int X, int Y,
                                           int rotation,
                                           int barcodeType,
                                           int barWidthRatio,
                                           int barcodeMultiplier,
                                           int barcodeHeight,
                                           int textUnder,
                                           byte[] barcodeData,
                                           out int err);

//use the function:

System.Text.ASCIIEncoding ascii = new ASCIIEncoding();
barcodeData = ascii.GetBytes("0123456789");

result = ZBRGDIDrawBarcode(X, Y, rotation, barcodeType, barWidthRatio,
   barcodeMultiplier, barcodeHeight, textUnder, barcodeData, out err);
```

5

Ethernet SmartCard Encoding SDK

Introduction

This section contains information for software developers intending to write applications for Zebra ZXP Series 3 Card Printers which require SmartCard encoding over Ethernet.

The Ethernet SmartCard Encoding Application Programming Interface (API) provides functions to perform Ethernet-based SmartCard encoding.

Functions

ZBRSXInitialize

Description: Initializes the Ethernet interface.

Syntax: int ZBRSXInitialize(out int error)

Parameters: [out] error code

Returns: error code (see Appendix A)

Sample Code: int alarm = ZBRSXInitialize(out error);

ZBRSXClose

Description: Closes the Ethernet interface.

Syntax: int ZBRSXClose(out int error)

Parameters: [out] error code

Returns: error code (see Appendix A)

Sample Code: int alarm = ZBRSXClose(out error);

ZBRSXDiscover

Description: Searches a printer for internal SmartCard readers.

Syntax:

```
int ZBRSXDiscover(
                        object       IPAddress,
                        out object   RetDevice,
                        out int      retError )
```

Parameters:

IPAddress	[in] string array containing IP address of printer.
RetDevice	[out] string array containing IP address of printer if SmartCard reader detected.
retError	[out]error code.

Returns: error code (see Appendix A)

Sample Code:

```
string[] ipAddr = new string[1];

//initialize array to printer's IP address
ipAddr[0] = "10.1.2.92";

object IPAddress = (object)ipAddr;

object RetDevice = null;

int retError = 0;

int alarm = ZBRSXDiscover(IPAddress, out RetDevice,
                          out retError);

if (alarm > 0)
    MessageBox.Show("ZBRSXDiscover returned error");

else if (RetDevice == null)
{
    MessageBox.Show("SmartCard Reader not found.");
}
else // SmartCard reader detected in printer
```

ZBRSXUSBEnum

Description: Identifies the printer and its associated SmartCard reader.

Syntax:
```
int ZBRSXUSBEnum(
                          object      IPAddress,
                          out object  RetDevice,
                          out int     retError )
```

Parameters:

IPAddress	[in]string variable containing the IP address of a printer.
RetDevice	[out]string array of printerId and its associated SmartCard readerId(s).
retError	[out]error code.

Returns: error code (see Appendix A)

Sample Code:
```
int alarm = ZBRSXUSBEnum(IPAddress, out RetDevice, out retError);
```

Note: Deprecated function - do not use. Use ZBRSXUSBEnumEx

ZBRSXUSBEnumEx

Description: Identifies the printer and its associated SmartCard reader.

Syntax:

```
int ZBRSXUSBEnumEx(
                        object      IPAddress,
                        IntPtr      devStruct,
                        int         devStructSize,
                        out int     structCount,
                        out int     retError )
```

Parameters:

IPAddress	[in]string variable containing the IP address of a printer.
devStruct	[out]array of structures containing the printerId and its associated SmartCard readerId(s). *Note that there will be exactly one structure in the array if the printer has an associated SmartCard reader.*
devStructSize	[in]size of devStruct.
structCount	[out]number of structures returned in devStruct. (Will be 1 if printer has a SmartCard reader.
retError	[out]error code.

Returns: error code (see Appendix A)

Sample Code:

```
int alarm = ZBRSXUSBEnumEx(IPAddress, out devStruct,
                           devStructSize, out structCount,
                           out retError);
```

Structure definition for devStruct parameter of ZBRSXUSBEnumEx:

```
[StructLayout(LayoutKind.Sequential)]
public struct DeviceStruct
{
    [MarshalAs(UnmanagedType.BStr)]
    public string printer;

    [MarshalAs(UnmanagedType.BStr)]
    public string reader;
}
```

ZBRSXConnect

Description: Opens a connection to a SmartCard reader within a printer.

Syntax:
```
int ZBRSXConnect(
                          string      deviceId,
                          bool        encrypt,
                          out int     retError )
```

Parameters:

deviceId	[in]SmartCard reader device ID.
Encrypt	[in] flag to indicate whether or not the connection will use encryption for smart card encoding. False = do not use encryption True = use encryption
retError	[out]error code.

Returns: error code (see Appendix A)

Sample Code:
```
int alarm = ZBRSXConnect(deviceId, encrypt, out retError);
```

ZBRSXDisconnect

Description: Closes a connection to a SmartCard reader within a printer.

Syntax:
```
int ZBRSXDisconnect(
                        string       deviceId,
                        out int     retError )
```

Parameters:

deviceId	[in]SmartCard reader device ID.
retError	[out]error code.

Returns: error code (see Appendix A)

Sample Code:
```
int alarm = ZBRSXDisconnect(deviceId, out retError);
```

ZBRSXGetPrinterName

Description: Returns the printer name for a device ID.

Syntax:
```
int ZBRSXGetPrinterName(
                          string      deviceId,
                          [MarshalAs(UnmanagedType.BStr)]
                          out string  deviceName,
                          out int     retError )
```

Parameters:

deviceId	[in]printer device ID.
deviceName	[out]printer name.
retError	[out]error code.

Returns: error code (see Appendix A)

Sample Code:
```
int alarm = ZBRSXGetPrinterName(deviceId, out deviceName,
                                  out retError);
```

ZBRSXGetPCSCReaderNames

Description: Returns the SmartCard reader name(s)associated with a SmartCard reader ID.

Syntax:

```
int ZBRSXGetPCSCReaderNames (
                        string      readerId,
                        out object  readerNames,
                        out int     error )
```

Parameters:

readerId	[in]SmartCard reader ID.
readerNames	[out]string array containing the SmartCard reader name(s).
error	[out]error code.

Returns: error code (see Appendix A)

Sample Code:

```
int alarm = ZBRSXGetPCSCReaderNames(readerId, out readerNames,
                                    out error);

if (readerNames != null)
{
    //get SmartCard reader name(s) found
    Array arr = (Array)readerNames;
    foreach (string reader in arr)
    {
        //contact & contactless reader names retrieved
    }
}
```

Note: The reader names returned in readerNames will be the SmartCard reader names required for opening/closing connections to contact & contactless SmartCards via PC/SC for SmartCard operations.

ZBRSXGetStatus

Description: Retrieves a printer's SmartCard reader's connection status.

Syntax:
```
int ZBRSXGetStatus (
                        string      deviceId,
                        out int     dwStatus,
                        out int     error )
```

Parameters:

deviceId	[in]	SmartCard reader device ID.
dwStatus	[out]	SmartCard reaeder's connection status: 1 = connection closed 2 = connection opened
error	[out]	error code.

Returns: error code (see Appendix A)

Sample Code:
```
int alarm = ZBRSXGetStatus(deviceId, out dwStatus, out error);
```

ZBRSXIsready

Description: Determines if a printer is in the ready state.

Syntax:
```
int ZBRSXIsready (
                        string      deviceId,
                        out long    dwStatus,
                        out int     error )
```

Parameters:

deviceId	[in]printer device ID.
dwStatus	[out]printer's current state: 0 = not ready 1 = ready
error	[out]error code.

Returns: error code (see Appendix A)

Sample Code:
```
int alarm = ZBRSXIsready(deviceId, out dwStatus, out error);
```

ZBRSXConfigureServer

Description: Sets the Printer's Ethernet configuration for specific IP address, subnet mask, and default gateway.

Syntax:

```
int ZBRSXConfigureServer (
                          string      currIPAddress,
                          byte[]      newIPAddress,
                          byte[]      newSubnet,
                          byte[]      newGateway,
                          out int     error )
```

Parameters:

currIPAddress	[in]Printer's current IP address.
newIPAddress	[in]New IP address for the Printer.
newSubnet	[in]New subnet mask for the Printer.
newGateway	[in]New default gateway for the Printer.
error	[out]error code.

Returns: error code (see Appendix A)

Sample Code:

```
int alarm = ZBRSXConfigureServer(currIPAddress, newIPAddress,
                                 newSubnet, newGateway,
                                 out error);
```

Note: The Printer will require rebooting after configuration.

ZBRSXRebootServer

Description: Performs a reboot of the the Printer.

Syntax:
```
int ZBRSXRebootServer (
                        string      deviceId,
                        out int     error )
```

Parameters:

deviceId	[in]printer device ID.
error	[out]error code.

Returns: error code (see Appendix A)

Sample Code:
```
int alarm = ZBRSXRebootServer(deviceId, out error);
```

Dlls required for distribution

The following dlls must be distributed with your application to support the Printer:

1. ZBRSXBridge.dll - contains the functions listed in this section.
2. Zbsconfigsrv.dll - contains low-level configuration functions.
3. Sxuptp.dll - contains low-level network protocol support.
4. _Setup.dll - contains installation support.
5. ZSCEncodeAP.dll - contains the functions listed in this section.

Note: A CD-ROM containing an installation program for the required dlls and device drivers is available from Zebra Technologies. This installation program can be included with your application's installer to simplify the installation and distribution process for Printer support.

6

Ethernet SmartCard Encoding Application Framework

Introduction

The Application Framework is a class which provides public methods to "wrap" each of the Ethernet SmartCard Encoding Application Programming Interface functions (APIs) and their required data structures. Additionally, the framework provides an interface facade.

The interface facade was developed for software developers who wish to write Ethernet-based SmartCard encoding applications but do not want to use the SDK directly; it facilitates rapid application development (RAD) for Ethernet-based SmartCard encoding solutions.

Data Structures

DeviceStruct

Description: This structure holds a printer deviceId and its associated SmartCard reader deviceId.

Definition:

```
[StructLayout(LayoutKind.Sequential)]
private struct DeviceStruct
{
    [MarshalAs(UnmanagedType.BStr)]
    public string printer;

    [MarshalAs(UnmanagedType.BStr)]
    public string reader;
}
```

Public Wrapper Methods

Initialize

Description: Calls ZBRSXClose to ensure the interface has no erroneous open connections, and then calls ZBRSXInitialize to initialize the interface.

Syntax: int Initialize(out int error)

Parameters: [out] error code

Returns: error code (see Appendix A)

Sample Code:
```
int alarm = Initialize(out error);
```

Close

Description: Calls ZBRSXClose to close the Ethernet interface.

Syntax: int Close(out int error)

Parameters: [out] error code

Returns: error code (see Appendix A)

Sample Code:
```
int alarm = Close(out error);
```

Discover

Description: Calls ZBRSXDiscover to search a printer for internal SmartCard readers.

Syntax:

```
int Discover(
                        object      IPAddress,
                        out object  RetDevice,
                        out int     retError )
```

Parameters:

IPAddress	[in] string array containing IP address of printer.
RetDevice	[out]string array containing IP address of printer if SmartCard reader detected.
retError	[out]error code.

Returns: error code (see Appendix A)

Sample Code:

```
string[] ipAddr = new string[1];

//initialize array to printer's IP address
ipAddr[0] = "10.1.2.92";

object IPAddress = (object)ipAddr;

object RetDevice = null;

int retError = 0;

int alarm = Discover(IPAddress, out RetDevice,
                     out retError);

if (alarm > 0)
    MessageBox.Show("Discover returned error");

else if (RetDevice == null)
{
    MessageBox.Show("SmartCard Reader not found.");
}
else // SmartCard reader detected in printer
```

USBEnum

Description: Calls ZBRSXUSBEnum to identify the printer and its associated SmartCard reader.

Syntax:
```
int USBEnum(
                        object      IPAddress,
                        out object  RetDevice,
                        out int     retError )
```

Parameters:

IPAddress	[in]string variable containing the IP address of a printer.
RetDevice	[out]string array of printerId and its associated SmartCard readerId(s).
retError	[out]error code.

Returns: error code (see Appendix A)

Sample Code:
```
int alarm = USBEnum(IPAddress, out RetDevice, out retError);
```

Note: Deprecated function - do not use. Use USBEnumEx

USBEnumEx

Description: Calls ZBRSXUSBEnumEx to identify the printer and its associated SmartCard reader.

Syntax:
```
int USBEnumEx(
                        object      IPAddress,
                        IntPtr      devStruct, int  devStructSize,
                        out int      structCount,
                        out int     retError )
```

Parameters:

IPAddress	[in]string variable containing the IP address of a printer.
devStruct	[out]array of structures containing the printerId and its associated SmartCard readerId(s). *Note that there will be exactly one structure in the array if the printer has an associated SmartCard reader.*
devStructSize	[in]size of devStruct.
structCount	[out]number of structures returned in devStruct. (Will be 1 if printer has a SmartCard reader.)
retError	[out]error code.

Returns: error code (see Appendix A)

Sample Code:
```
int alarm = USBEnumEx(IPAddress, out devStruct, devStructSize,
     out structCount, out retError);
```

Note: See DevStruct definition for details regarding devStruct parameter.

Connect

Description: Calls ZBRSXConnect to open a connection to a SmartCard reader within a printer.

Syntax:
```
int Connect(
                        string      deviceId,
                        bool        encrypt,
                        out int     retError )
```

Parameters:

deviceId	[in]SmartCard reader device ID.
Encrypt	[in]flag to indicate whether or not the connection will use encryption for SmartCard encoding. False = do not use encryption True = use encryption
retError	[out]error code.

Returns: error code (see Appendix A)

Sample Code:
```
int alarm = Connect(deviceId, encrypt, out retError);
```

Disconnect

Description: Calls ZBRSXDisconnect to close a connection to a SmartCard reader within a printer.

Syntax:

```
int Disconnect(
                              string      deviceId,
                              out int     retError )
```

Parameters:

deviceId	[in]SmartCard reader device ID.
retError	[out]error code.

Returns: error code (see Appendix A)

Sample Code:

```
int alarm = Disconnect(deviceId, out retError);
```

GetPrinterName

Description: Calls ZBRSXGetPrinterName to return the printer name for a device ID.

Syntax:

```
int GetPrinterName(
                        string      deviceId,
                        [MarshalAs(UnmanagedType.BStr)]
                        out string  deviceName,
                        out int     retError )
```

Parameters:

deviceId	[in]printer device ID.
deviceName	[out]printer name.
retError	[out]error code.

Returns: error code (see Appendix A)

Sample Code:

```
int alarm = GetPrinterName(deviceId, out deviceName,
                                    out retError);
```

GetPCSCReaderNames

Description: Provides a simpler interface for calling ZBRSXGetPCSCReaderNames to return the SmartCard reader name(s)associated with a SmartCard reader ID.

Syntax:
```
bool GetPCSCReaderNames (
                        string      readerId,
                        out string  contactlessName,
                        out string  contactName,
                        out string  errMsg )
```

Parameters:

readerId	[in]SmartCard reader ID.
contactlessName	[out]string containing the contactless smart card reader name.
contactName	[out]string containing the contact smart card reader name.
errMsg	[out]error message.

Returns: true = SmartCard reader names retrieved
false = SmartCard reader names were not found

Sample Code:
```
int alarm = GetPCSCReaderNames(readerId, out contactlessName,
                                    out contactName, out errMsg);
```

Note: The reader names returned in contactlessName & contactName will be the SmartCard reader names required for opening/closing connections to the contact & contactless SmartCards via the PC/SC interface for SmartCard operations.

GetStatus

Description: Calls ZBRSXGetStatus to retrieve a printer's SmartCard reader's connection status.

Syntax:
```
int GetStatus (
                        string      deviceId,
                        out int     dwStatus,
                        out int     error )
```

Parameters:

deviceId	[in]SmartCard reader device ID.
dwStatus	[out]SmartCard reader's connection status: 1 = connection closed 2 = connection opened
error	[out]error code.

Returns: error code (see Appendix A)

Sample Code:
```
int alarm = GetStatus(deviceId, out dwStatus, out error);
```

Isready

Description: Calls ZBRSXIsReady to determine if a printer is in the ready state.

Syntax:

```
int Isready (
                        string      deviceId,
                        out long    dwStatus,
                        out int     error )
```

Parameters:

deviceId	[in]printer device ID.
dwStatus	[out]printer's current state: 0 = not ready 1 = ready
error	[out]error code.

Returns: error code (see Appendix A)

Sample Code:

```
int alarm = Isready(deviceId, out dwStatus, out error);
```

ConfigureServer

Description: Provides a simpler interface for calling ZBRSXConfigureServer to configure the Printer for specific IP address, subnet mask, and default gateway.

Syntax:

```
int ConfigureServer (
                      string      currIPAddress,
                      string      newIPAddress,
                      string      newSubnet,
                      string      newGateway,
                      out string  errMsg )
```

Parameters:

currIPAddress	[in]Printer's current IP address.
newIPAddress	[in]New IP address for the Printer.
newSubnet	[in]New subnet mask for the Printer.
newGateway	[in]New default gateway for the Printer.
errMsg	[out]error message if error encountered.

Returns: error code (see Appendix A)

Sample Code:

```
int alarm = ConfigureServer(currIPAddress, newIPAddress,
                                  newSubnet, newGateway,out
                                  out errMsg);
```

Note: The Printer will require rebooting after configuration.

RebootServer

Description: Performs a reboot of the Printer.

Syntax:
```
int RebootServer (
                        string      deviceId,
                        out int     error )
```

Parameters:

deviceId	[in]printer device ID.
error	[out]error code.

Returns: error code (see Appendix A)

Sample Code:
```
int alarm = RebootServer(deviceId, out error);
```

Interface Facade

The interface facade provides a simpler interface, as an alternative to using the Ethernet SmartCard Encoding SDK directly, for performing Ethernet-based smartcard encoding. This section describes the interface facade.

For programming details, see Interface Facade Sample Code on page 210.

Methods

GetSmartCardReaderNames

```
Description: Locates a printer's SmartCard readers, retrieves their Names and
             returns the SmartCard reader Id.

Syntax:      bool GetSmartCardReaderNames (
                                string[]     IpAddress,
                                out string   contactlessName,
                                out string   contactName,
                                out string   smartCardReaderId,
                                out string   errMsg )

Parameters:  IpAddress           [in] 1-dimenstional string array containing
                                  the printer's IP Address.
             contactlessName     [out] contactless smartcard reader name
             contactName         [out] contact smartcard reader name
             smartCardReaderId   [out] smartcard reader Id
             errMsg              [out]error message if error encountered.

Returns:     true = smartcard reader names retrieved.
             false = smartcard readers not found.

Sample Code: bool result = GetSmartCardReaderNames(IpAddress,
                                            out contactlessName,
                                            out contactName,
                                            out smartCardReaderId,
                                            out errMsg);

Note:        If this function succeeds, the property SmartCardReaderNameId
             will be set to the smartcard reader Id, and the names returned in
             the contactlessName and contactName variables will be the
             smartcard reader names required by the PC/SC interface.
```

ConnectToReader

Description: Opens a connection to a printer's SmartCard reader.

Syntax:
```
bool ConnectToReader(
                        string[]    IpAddress,
                        string      smartCardReaderId,
                        bool        encrypt,
                        out         string errMsg )
```

Parameters:

IpAddress	[in] 1-dimensional string array containing the printer's IP Address.
smartCardReaderId	[in] string variable containing the device identifier for the smartcard module (returned by GetSmartCardReaderNames)
Encrypt	[in] flag indicating whether or not the connection should include encrypting the data passed betweenthe host and device. true = include encryption false = exclude encryption
errMsg	[out] error message if error encountered.

Returns: true = connection opened.
false = failed to open connection.

Sample Code:
```
string IpAddress = "127.0.0.1"
string smartCardReaderId;   //value returned from GetSmartCardReaderNames
bool encrypt = true;        // use encryption
string errMsg = string.Empty;

bool result = ConnectToReader(IpAddress, smartCardReaderId, encrypt,
                              out errMsg);
```

DisconnectFromReader

Description: Closes a connection to a printer's SmartCard reader.

Syntax:
```
bool DisconnectFromReader(
                        out string  errMsg )
```

Parameters:

errMsg	[out]error message if error encountered.

Returns:
```
true = connection closed.
false = failed to close connection.
```

Sample Code:
```
bool result = DisconnectFromReader(out errMsg);
```

Dlls required for distribution

The following dlls must be distributed with your application to support the Printer:

1. ZXRSXBridge.dll - contains the functions listed in this section.
2. Zbsconfigsrv.dll - contains low-level configuration functions.
3. Sxuptp.dll - contains low-level network protocol support.
4. _Setup.dll - contains installation support.

Note: A CD-ROM containing an installation program for the required dlls and device drivers is available from Zebra Technologies. This installation program can be included with your application's installer to simplify the installation and distribution process for Printer support.

7

Programming Examples

The programming examples in this section show how to use Zebra SDK functions and Windows API to perform printer-specific operations:

Basic Card Printing and Magnetic Stripe Encoding

The following example shows how to encode three tracks of magnetic stripe data, and then print an image and text.

```
// ZBRPrinter.dll and ZBRGraphics.dll C# Example
//*************************************************************************
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;
using System.IO;
using System.Drawing;

namespace BasicPrintMagExample
{
  class Program
  {
  //ZBRPrinter.dll imports

  //Handle functions
  [DllImport("ZBRPrinter.dll", EntryPoint="ZBRGetHandle",
  CharSet=CharSet.Auto, SetLastError=true)]
  public static extern int ZBRGetHandle(out IntPtr handle, byte[] drvName,
                                        out int printerType, out int err);

  [DllImport("ZBRPrinter.dll", EntryPoint="ZBRCloseHandle",
  CharSet=CharSet.Auto, SetLastError=true)]
  public static extern int ZBRCloseHandle(IntPtr handle, out int err);

  //Position functions
  [DllImport("ZBRPrinter.dll", EntryPoint = "ZBRPRNMovePrintReady",
  CharSet = CharSet.Auto, SetLastError = true)]
  public static extern int ZBRPRNMovePrintReady(IntPtr handle,
                                                int printerType,
                                                out int err);

  //Magnetic encoder functions
  [DllImport("ZBRPrinter.dll", EntryPoint = "ZBRPRNReadMag",
  CharSet = CharSet.Auto, SetLastError = true)]
  public static extern int ZBRPRNReadMag(IntPtr handle, int printerType,
                                         int trksToRead,
                                         byte[] trk1Buf,
                                         out int trk1BytesNeeded,
                                         byte[] trk2Buf,
                                         out int trk2BytesNeeded,
                                         byte[] trk3Buf,
                                         out int trk3BytesNeeded,
                                        out int err);

  [DllImport( "ZBRPrinter.dll", EntryPoint="ZBRPRNWriteMag",
  CharSet=CharSet.Auto, SetLastError=true)]
  public static extern int ZBRPRNWriteMag(IntPtr handle, int printerType,
                                          int trksToWrite, byte[] trk1Data,
                                          byte[] trk2Data, byte[] trk3Data,
                                          out int err);
```

```
//ZBRGraphics.dll imports

//Graphics buffer functions
[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIInitGraphics",
CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIInitGraphics(byte[] strPrinterName,
                                            out IntPtr hDC,
                                            out int err);

[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIPrintGraphics",
CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDIPrintGraphics(IntPtr hDC, out int err);

[DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDICloseGraphics",
CharSet=CharSet.Auto, SetLastError=true)]
public static extern int ZBRGDICloseGraphics(IntPtr hDC, out int err);

//Draw functions
[DllImport("ZBRGraphics.dll", EntryPoint = "ZBRGDIDrawImageRect",
CharSet = CharSet.Auto, SetLastError = true)]
public static extern int ZBRGDIDrawImageRect(byte[] fileName, int x, int y,
                                             int width, int height,
                                             out int err);

[DllImport("ZBRGraphics.dll", EntryPoint = "ZBRGDIDrawText",
CharSet = CharSet.Auto, SetLastError = true)]
public static extern int ZBRGDIDrawText(int x, int y,
                                        byte[] text, byte[] font,
                                        int fontSize, int fontStyle,
                                        int color, out int err);

//Utility Functions:
public static byte[] ImageToByteArray(string filename)
{
  MemoryStream ms = new MemoryStream();
  Image img = System.Drawing.Image.FromFile(filename);

  img.Save(ms, System.Drawing.Imaging.ImageFormat.Bmp);

  byte[] arr = ms.ToArray();

  ms = null;

  return arr;
}
```

```
//Program entry point:

static void Main(string[] args)
{
  //Printer SDK example:

  //Type Defines for ZBRPrinter.dll functions

  IntPtr  handle = IntPtr.Zero; //printer driver handle

  int     printerType = 0; //printer type

  int     err = 0;
  string printerName = "Zebra ZXP Series 3 USB Printer";

  //get handle to printer driver:

  byte[] prnDriver = ascil.GetBytes(printerName);

  int result = ZBRGetHandle(out handle, prnDriver, out printerType, out err);

  if( (result != 1) ||(err > 0)) //error getting handle
  {
     ascil = null;
     return 0; //exit
  }

  //-------------------------------------------------------------------------
  // the following logic is used to select the track(s) to magnetically encode:
  // 1 = encode track 1 only
  // 2 = encode track 2 only
  // 4 = encode track 3 only
  // by OR'ing the above values appropriately, multiple tracks can be encoded.
  // example: 7 = encode all 3 tracks (logic OR of 1, 2, and 4)
  //-------------------------------------------------------------------------

  //magnetic encode all 3 tracks of the card:
  int tracksToEncode = 7;

  string track1Data = "ABCDEFGHIJKLMNOPQRS";
  string track2Data = "0123456789987654321";
  string track3Data = "11223344556677889900";

  byte[] trk1Data = ascil.GetBytes(track1Data);
  byte[] trk2Data = ascil.GetBytes(track2Data);
  byte[] trk3Data = ascil.GetBytes(track3Data);

  err = 0;

  result = ZBRPRNWriteMag(handle, printerType, tracksToEncode,
                          trk1Data, trk2Data, trk3Data, out err);

  if ((result != 1) || (err > 0)) //error magnetic encoding
  {
     ascil = null;
     ZBRCloseHandle(handle, out err);
     return; //exit
  }
```

```
//read back the data from all 3 tracks:
int tracksToRead = 7;

byte[] trk1Buff = null;
byte[] trk2Buff = null;
byte[] trk3Buff = null;

int trk1BuffSize = 0;
int trk2BuffSize = 0;
int trk3BuffSize = 0;


result = ZBRPRNReadMag(handle, printerType, tracksToRead,
                       trk1Buff, out trk1BuffSize,
                       trk2Buff, out trk2BuffSize,
                       trk3Buff, out trk3BuffSize, out err);

if ((result != 1) || (err > 0)) //error reading magnetic data
{
    ascii = null;
    ZBRCloseHandle(handle, out err);
    return; //exit
}

//compare the data written and read:
for (int i = 0; i < trk1BuffSize; i++)
{
    if (trk1Data[i] != trk1Buff[i])
    {
           //error
           ascii = null;
           ZBRCloseHandle(handle, out err);
           return;
    }
}

for (int i = 0; i < trk2BuffSize; i++)
{
    if (trk2Data[i] != trk2Buff[i])
    {
           //error
           ascii = null;
           ZBRCloseHandle(handle, out err);
    return;
    }
}

for (int i = 0; i < trk3BuffSize; i++)
{
    if (trk3Data[i] != trk3Buff[i])
    {
           //error
           ascii = null;
           ZBRCloseHandle(handle, out err);
           return;
    }
}
```

```
//move card to print ready position:

result = ZBRPRNMovePrintReady(handle, printerType, out err);

if ((result != 1) || (err > 0)) //error positioning card
{
    ascii = null;
    ZBRCloseHandle(handle, out err);
    return; //exit
}

//close handle to printer driver
result = ZBRCloseHandle(handle, out err);

if ((result != 1) || (err > 0)) //error closing handle
{
    ascii = null;
    return; //exit
}

//-----------------------------------------------------
//Printer SDK tasks completed
//-----------------------------------------------------
```

```
//Graphics SDK example:

//Type Defines for ZBRPrinter.dll functions

IntPtr hDC = IntPtr.Zero; //printer device context handle

//Initialize the grpahics buffer:
result = ZBRGDIInitGraphics(prnDriver, out hDC, out err);

if ((result != 1) || (err > 0)) //error initializing buffer
{
    return; //exit
}

//Draw an image into the buffer:
byte[] image = ImageToByteArray("Zebra.bmp");
int xCoord = 50;
int yCoord = 50;
int width = 200;
int height = 150;

result = ZBRGDIDrawImageRect(image, xCoord, yCoord, width, height, out err);

if ((result != 1) || (err > 0)) //error drawing image into buffer
{
    ascii = null;
    ZBRGDICloseGraphics(hDC, out err);
    return; //exit
}

//Draw text in the buffer:
byte[] text = ascii.GetBytes("Text written to card");
byte[] font = ascii.GetBytes("Arial");
int fontSize = 12;
int fontBold = 0x01;
int fontColor = 0xFF0000; //black

xCoord = 250;
yCoord = 250;

result = ZBRGDIDrawText(xCoord, yCoord, text, font, fontSize, fontBold,
                        mfontColor, out err);

if ((result != 1) || (err > 0)) //error drawing text into buffer
{
    ascii = null;
    ZBRGDICloseGraphics(hDC, out err);
    return; //exit
}

//print image and text in graphics buffer:
result = ZBRGDIPrintGraphics(hDC, out err);
if ((result != 1) || (err > 0)) //error drawing text into buffer
{
}
```

```
    //close graphics buffer:
    result = ZBRGDICloseGraphics(hDC, out err);
    if ((result != 1) || (err > 0)) //error closing buffer
    {
    }

    ascii = null;

    //-----------------------------------------------------
    //Graphics SDK tasks completed
    //-----------------------------------------------------

    }
  }
}
```

Contact SmartCard USB Connection

The following example demonstrates the following:

1. How to position a SLE 4442 SmartCard for encoding via USB connection.
2. How to position the card for printing if printing is to be done after encoding.
3. How to eject the card from the printer no further processing is required after SmartCard encoding.

```
//Contact SmartCard encoding C# example

using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;
using System.IO;
using System.Drawing;

namespace BasicContactSmartCardExample
{
class Program
  {
  //ZBRPrinter.dll imports

  //Handle functions
  [DllImport("ZBRPrinter.dll", EntryPoint = "ZBRGetHandle",
  CharSet = CharSet.Auto, SetLastError = true)]
  public static extern int ZBRGetHandle(out IntPtr handle, byte[] drvName,
                                        out int printerType, out int err);

  [DllImport("ZBRPrinter.dll", EntryPoint = "ZBRCloseHandle",
  CharSet = CharSet.Auto, SetLastError = true)]
  public static extern int ZBRCloseHandle(IntPtr handle, out int err);

  //Card movement functions
  [DllImport("ZBRPrinter.dll", EntryPoint = "ZBRPRNEjectCard",
  CharSet = CharSet.Auto, SetLastError = true)]
  public static extern int ZBRPRNEjectCard(IntPtr _handle, int prn_type, out int err);

  //SmartCard functions
  [DllImport("ZBRPrinter.dll", EntryPoint = "ZBRPRNStartSmartCard",
  CharSet = CharSet.Auto, SetLastError = true)]
  public static extern int ZBRPRNStartSmartCard(IntPtr _handle,
  int printerType, int cardType, out int err);

  [DllImport("ZBRPrinter.dll", EntryPoint = "ZBRPRNEndSmartCard",
  CharSet = CharSet.Auto, SetLastError = true)]
  public static extern int ZBRPRNEndSmartCard(IntPtr _handle, int printerType, int cardType,
                                              int movement, out int err);
```

```
static void Main(string[] args)
{
    ASCIIEncoding ascii = new ASCIIEncoding();

    //Type Defines for ZBRPrinter.dll functions
    const int CONTACT_CARD = 0x01;
    const int MOVE_CARD_TO_PRINT_READY = 0x00;

    IntPtr handle = IntPtr.Zero;
    int printerType = 0;
    int err = 0;
    string printerName = "Zebra ZXP Series 3 USB Printer";

    //Get handle to printer:
    byte[] prnDriver = ascii.GetBytes(printerName);

    int result = ZBRGetHandle(out handle, prnDriver, out printerType, out err);

    if ((result != 1) || (err > 0)) //error getting handle
    {
           ascii = null;
           return; //exit
    }

    //position the card at the contact SmartCard encoder:
    result = ZBRPRNStartSmartCard(handle, printerType, CONTACT_CARD, out err);

    if ((result != 1) || (err > 0)) //error positioning card for encoding
    {
           ascii = null;

           //eject the card from the printer:
           result = ZBRPRNEjectCard(handle, printerType, out err);
           if ((result != 1) || (err > 0)) //error ejecting card
           {
           }

           ZBRCloseHandle(handle, out err);
           return; //exit
    }

    //-----------------------------------------------
    //Place PC/SC Encoding here:
    //-----------------------------------------------
```

```
        //Psuedo-code for post-SmartCard processing
        if (smartCardEncoding == failed)
        {
               ascii = null;

               //eject the card from the printer:
               result = ZBRPRNEjectCard(handle, printerType, out err);
               if ((result != 1) || (err > 0)) //error ejecting card
               {
               }

               ZBRCloseHandle(handle, out err);
               return; //exit
        }
        else //SmartCard encoding succeeded
        {
               if (EncodingOnly == true)
               {
                      ascii = null;

                      //eject the card from the printer:
                      result = ZBRPRNEjectCard(handle, printerType, out err);
                      if ((result != 1) || (err > 0)) //error ejecting card
                      {
                      }

                      ZBRCloseHandle(handle, out err);
                      return;
               }
               else //position card for printing
               {
                      result = ZBRPRNEndSmartCard(handle, printerType,
                                                  CONTACT_CARD,
                                                  MOVE_CARD_TO_PRINT_READY,
                                                  out err);

                      if ((result != 1) || (err > 0))     //error moving card to print
                                                          //ready position
                      {
                             ascii = null;

                             ZBRCloseHandle(handle, out err);
                             return;
                      }
               }
        }

        //--------------------------------
        //Perform Printing As Required
        //--------------------------------

        //Post-printing cleanup:
        ascii = null;
        ZBRCloseHandle(handle, out err);
    }
  }
}
```

Contactless SmartCard USB Connection

The following example demonstrates the following:

1. How to position a MIFARE 1K SmartCard for encoding via USB connection.
2. How to position the card for printing if printing is to be done after encoding.
3. How to eject the card from the printer no further processing is required after SmartCard encoding.

```
//Contactless SmartCard encoding C# example
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;
using System.IO;
using System.Drawing;

namespace BasicContactlessSmartCardExample
{
class Program
  {
  //ZBRPrinter.dll imports

  //Handle functions
  [DllImport("ZBRPrinter.dll", EntryPoint = "ZBRGetHandle",
  CharSet = CharSet.Auto, SetLastError = true)]
  public static extern int ZBRGetHandle(out IntPtr handle, byte[] drvName,
                                        out int printerType, out int err);

  [DllImport("ZBRPrinter.dll", EntryPoint = "ZBRCloseHandle",
  CharSet = CharSet.Auto, SetLastError = true)]
  public static extern int ZBRCloseHandle(IntPtr handle, out int err);

  //Card movement functions
  [DllImport("ZBRPrinter.dll", EntryPoint = "ZBRPRNEjectCard",
  CharSet = CharSet.Auto, SetLastError = true)]
  public static extern int ZBRPRNEjectCard(IntPtr _handle, int prn_type, out int err);

  //SmartCard functions
  [DllImport("ZBRPrinter.dll", EntryPoint = "ZBRPRNStartSmartCard",
  CharSet = CharSet.Auto, SetLastError = true)]
  public static extern int ZBRPRNStartSmartCard(IntPtr _handle,
  int printerType, int cardType, out int err);

  [DllImport("ZBRPrinter.dll", EntryPoint = "ZBRPRNEndSmartCard",
  CharSet = CharSet.Auto, SetLastError = true)]
  public static extern int ZBRPRNEndSmartCard(IntPtr _handle, int printerType,
                                              int cardType, int movement, out int err);
```

```
static void Main(string[] args)
{
    ASCIIEncoding ascii = new ASCIIEncoding();

    //Type Defines for ZBRPrinter.dll functions
    const int CONTACTLESS_CARD = 0x02;
    const int MOVE_CARD_TO_PRINT_READY = 0x00;

    IntPtr handle = IntPtr.Zero;
    int printerType = 0;
    int err = 0;
    string printerName = "Zebra ZXP Series 3 USB Printer";

    //Get handle to printer:
    byte[] prnDriver = ascii.GetBytes(printerName);

    int result = ZBRGetHandle(out handle, prnDriver, out printerType, out err);
    if ((result != 1) || (err > 0)) //error getting handle
    {
           ascii = null;
           return; //exit
    }

    //position the card at the contactless SmartCard encoder:
    result = ZBRPRNStartSmartCard(handle, printerType, CONTACTLESS_CARD, out err);
    if ((result != 1) || (err > 0)) //error positioning card for encoding
    {
           ascii = null;

           //eject the card from the printer:
           result = ZBRPRNEjectCard(handle, printerType, out err);
           if ((result != 1) || (err > 0)) //error ejecting card
           {
           }

           ZBRCloseHandle(handle, out err);
           return; //exit
    }


    //-----------------------------------------------
    //Place PC/SC Encoding here:
    //-----------------------------------------------
```

```
        //Psuedo-code for post SmartCard processing
        if (smartCardEncoding == failed)
        {
                ascii = null;

                //eject the card from the printer:
                result = ZBRPRNEjectCard(handle, printerType, out err);
                if ((result != 1) || (err > 0)) //error ejecting card
                {
                }

                ZBRCloseHandle(handle, out err);
                return; //exit
        }
        else //SmartCard encoding succeeded
        {
                if (EncodingOnly == true)
                {
                        ascii = null;

                        //eject the card from the printer:
                        result = ZBRPRNEjectCard(handle, printerType, out err);
                        if ((result != 1) || (err > 0)) //error ejecting card
                        {
                        }

                        ZBRCloseHandle(handle, out err);
                        return;
                }
                else //position card for printing
                {
                        result = ZBRPRNEndSmartCard(handle, printerType,
                                                    CONTACTLESS_CARD,
                                                    MOVE_CARD_TO_PRINT_READY,
                                                    out err);
                        if ((result != 1) || (err > 0))     //error moving card to print
                                                            //ready position
                        {
                                ascii = null;

                                ZBRCloseHandle(handle, out err);
                                return;
                        }
                }
        }

        //---------------------------------
        //Perform Printing As Required
        //---------------------------------

        //Post-printing cleanup:
        ascii = null;
        ZBRCloseHandle(handle, out err);
    }
  }
}
```

Barcode

The following example shows how to print an image, text, and a barcode on a card.

```
//ZBRGraphics.dll C# Example
//************************************************************************
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;
using System.IO;
using System.Drawing;

namespace BasicBarcodeExample
{
class Program
  {
  //ZBRGraphics.dll imports
  //Graphics buffer functions
  [DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIInitGraphics",
  CharSet=CharSet.Auto, SetLastError=true)]
  public static extern int ZBRGDIInitGraphics(byte[] strPrinterName,
                                              out IntPtr hDC,
                                              out int err);

  [DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDIPrintGraphics",
  CharSet=CharSet.Auto, SetLastError=true)]
  public static extern int ZBRGDIPrintGraphics(IntPtr hDC, out int err);
  [DllImport("ZBRGraphics.dll",EntryPoint="ZBRGDICloseGraphics",
  CharSet=CharSet.Auto, SetLastError=true)]
  public static extern int ZBRGDICloseGraphics(IntPtr hDC, out int err);

  //Draw functions
  [DllImport("ZBRGraphics.dll", EntryPoint = "ZBRGDIDrawImageRect",
  CharSet = CharSet.Auto, SetLastError = true)]
  public static extern int ZBRGDIDrawImageRect(byte[] fileName, int x, int y, int width,
                                               int height, out int err);

  [DllImport("ZBRGraphics.dll", EntryPoint = "ZBRGDIDrawText",
  CharSet = CharSet.Auto, SetLastError = true)]
  public static extern int ZBRGDIDrawText(int x, int y, byte[] text, byte[] font,
                                          int fontSize, int fontStyle,
                                          int color, out int err);

  [DllImport("ZBRGraphics.dll", EntryPoint = "ZBRGDIDrawBarCode",
  CharSet = CharSet.Auto, SetLastError = true)]
  public static extern int ZBRGDIDrawBarCode(int x, int y, int rotation, int barcodeType,
                                             int widthRatio, int multiplier, int height,
                                             int textUnder, byte[] data, out int err);

  //Utility Functions:
  public static byte[] ImageToByteArray(string filename)
  {
    MemoryStream ms = new MemoryStream();
    Image img = System.Drawing.Image.FromFile(filename);

    img.Save(ms, System.Drawing.Imaging.ImageFormat.Bmp);

    byte[] arr = ms.ToArray();
    ms = null;
    return arr;
  }
```

```
//Program entry point:
static void Main(string[] args)
{
    ASCIIEncoding ascii = new ASCIIEncoding();

    //Type Defines for ZBRPrinter.dll functions

    IntPtr hDC = IntPtr.Zero; //printer device context handle

    string printerName = "Zebra ZXP Series 3 USB Printer";
    int err = 0;
    int result = 0;
    byte[] prnDriver = ascii.GetBytes(printerName);

    //Initialize the grpahics buffer:
    result = ZBRGDIInitGraphics(prnDriver, out hDC, out err);

    if ((result != 1) || (err > 0)) //error initializing buffer
    {
           return; //exit
    }

    //Draw an image into the buffer:
    byte[] image = ImageToByteArray("Zebra.bmp");

    int xCoord = 50;
    int yCoord = 50;
    int width = 200;
    int height = 150;

    result = ZBRGDIDrawImageRect(image, xCoord, yCoord, width, height, out err);

    if ((result != 1) || (err > 0)) //error drawing image into buffer
    {
           ascii = null;
           ZBRGDICloseGraphics(hDC, out err);
           return; //exit
    }
```

```
        //Draw text in the buffer:
        byte[] text = ascii.GetBytes("Text written to card");
        byte[] font = ascii.GetBytes("Arial");
        int fontSize = 12;
        int fontBold = 0x01;
        int fontColor = 0xFF0000; //black

        xCoord = 250;
        yCoord = 250;

        result = ZBRGDIDrawText(xCoord, yCoord, text, font, fontSize, fontBold,
                                fontColor, out err);

        if ((result != 1) || (err > 0)) //error drawing text into buffer
        {
               ascii = null;
               ZBRGDICloseGraphics(hDC, out err);
               return; //exit
        }

        //Draw a barcode into monochrome image buffer:
        int startX = 280;
        int startY = 590;
        int rotation = 0;    //no rotation
        int barcodeType = 0; //Code 39
        int widthRatio = 2;  //narrow bar = 2 dots, wide bar = 5 dots
        int multiplier = 2; //(2..9)
        int height = 50;    //50 dots
        int textUnder = 1;  // true
        string barcodeData = "1234567890";
        byte[]data = ascii.GetBytes(barcodeData);

        err = 0;

        result = ZBRGDIDrawBarCode(startX, startY, rotation, barcodeType, widthRatio,
                                   multiplier, height, textUnder, data, out err);

        if ((result != 1) || (err > 0)) //error drawing barcode into buffer
        {
               ascii = null;
               ZBRGDICloseGraphics(hDC, out err);
               return; //exit
        }

        //print image, text, and barcode in graphics buffer:
        result = ZBRGDIPrintGraphics(hDC, out err);
        if ((result != 1) || (err > 0)) //error printing buffer
        {
        }

        //close graphics buffer:
        result = ZBRGDICloseGraphics(hDC, out err);
        if ((result != 1) || (err > 0)) //error closing buffer
        {
        }

        ascii = null;

    }
  }
}
```

Interface Facade Sample Code

(for SmartCard Encoding via an Ethernet Connection)

The following example showsthe usage of the interface facade to perform Ethernet-based SmartCard encoding.

```
//This function demonstrates how to retrieve a printer's SmartCard reader names & the
//SmartCard reader Id.
private void LocateReaders(out string contactlessReader,
                           out string contactReader,
                           out string smartCardReaderId)
{
       string[] IpAddress = null;
       SCEncodingEthernetAppFramework SCEncodeEthernet = null;
       contactlessReader = string.Empty;
       contactReader = string.Empty;
       smartCardReaderId = string.Empty;

       try
       {
              SCEncodeEthernet = new SCEncodingEthernetAppFramework();
              string errMsg = string.Empty;
              IpAddress = new string[1];
              IpAddress[0] = "10.1.5.75"; //printer IP address

              if (!SCEncodingEthernet.GetSmartCardReaderNames(IpAddress,
                                                  out _ScontactlessReader,
                                                  out _ScontactReader,
                                                  out _SsmartCardReaderId,
                                                  out errMsg) )
              {
                     MessageBox.Show("Failed to locate SmartCard readers: " + errMsg);
              }
       }
       catch (Exception ex)
       {
              MessageBox.Show(ex.Message, "LocateReaders Exception");
       }
       finally
       {
              SCEncodeEthernet = null;
              IpAddress = null;
       }
}
```

```
//This function demonstrates how to open a connection to the SmartCard reader, where to
//place the PC/SC SmartCard encoding logic, and close the/connection to the reader.
private void PerformSmartCardEncoding(string smartCardReaderId,
                                      string contactlessReader,
                                      string contactReader)
{
       string[] IpAddress = null;
       SCEncodingEthernetAppFramework SCEncodeEthernet = null;
       try
       {
              SCEncodeEthernet = new SCEncodingEthernetAppFramework();
              string errMsg = string.Empty;
              IpAddress = new string[1];
              IpAddress[0] = "10.1.5.75"; //printer IP address

              //assign the SmartCard reader id:
              SCEncodeEthernet.SmartCardReaderId = smartCardReaderId;

              //connect to the reader without encrytion:
              if (SCEncodeEthernet.ConnectToReader(IpAddress, smartCardReaderId, false,
                                                   out errMsg))
              {
                     //place PC/SC SmartCard encoding logic here

                     //close the reader connection:
                     SCEncodeEthernet.DisconnectFromReader(out errMsg);
              }
              else
              {
              MessageBox.Show("Failed to connect to SmartCard reader: " + errMsg);
              }
       }
       catch (Exception ex)
       {
              MessageBox.Show(ex.Message, "PerformSmartCardEncoding Exception");
       }
       finally
       {
              SCEncodeEthernet = null;
              IpAddress = null;
       }
}
```

Appendix A

Error Codes

This appendix lists error codes, error messages, and possible causes for all error messages that may appear when running applications created with the SDK for Zebra ZXP Series 1 and ZXP Series 3 card printers.

ZXP Series 3 Specific Error Codes

CODE	ERROR	POSSIBLE CAUSE
-1	ZBR_ERROR_PRINTER_MECHANICAL_ERROR	Mechanical error
1	ZBR_ERROR_BROKEN_RIBBON	Indicates a broken ribbon
2	ZBR_ERROR_TEMPERATURE	Print head temperature is too high
3	ZBR_ERROR_MECHANICAL_ERROR	Mechanical error
4	ZBR_ERROR_OUT_OF_CARD	Printer is out of cards, or unable to feed the card
5	ZBR_ERROR_CARD_IN_ENCODER	Unable to encode magnetic or smart card encoder
6	ZBR_ERROR_CARD_NOT_IN_ENCODER	Unable to encode the card because it is not in the encoder
7	ZBR_ERROR_PRINT_HEAD_OPEN	Print head is up
8	ZBR_ERROR_OUT_OF_RIBBON	Out of ribbon
9	ZBR_ERROR_REMOVE_RIBBON	Ribbon needs to be removed
10	ZBR_ERROR_PARAMETERS_ERROR	Wrong number of parameters or a value is incorrect
11	ZBR_ERROR_INVALID_COORDINATES	Invalid coordinates while trying to draw a barcode or graphics
12	ZBR_ERROR_UNKNOWN_BARCODE	Undefined barcode type
13	ZBR_ERROR_UNKNOWN_TEXT	Text for magnetic encoding or bar code drawing is invalid
14	ZBR_ERROR_COMMAND_ERROR	Invalid command
20	ZBR_ERROR_BARCODE_DATA_SYNTAX	Syntax error in the barcode command or parameters
21	ZBR_ERROR_TEXT_DATA_SYNTAX	General text data error
22	ZBR_ERROR_GRAPHIC_DATA_SYNTAX	Syntax error in the graphic command data
30	ZBR_ERROR_GRAPHIC_IMAGE_INITIALIZATION	Unable to initialize the graphics buffer
31	ZBR_ERROR_GRAPHIC_IMAGE_MAXIMUM_WIDTH_EXCEEDED	Graphic object to be drawn exceeds the X range
32	ZBR_ERROR_GRAPHIC_IMAGE_MAXIMUM_HEIGHT_EXCEEDED	Graphic object to be drawn exceeds the Y range
33	ZBR_ERROR_GRAPHIC_IMAGE_DATA_CHECKSUM_ERROR	Graphic data checksum error

CODE	ERROR	POSSIBLE CAUSE
34	ZBR_ERROR_DATA_TRANSFER_TIME_OUT	Data time-out error, usually happens when the USB cable is taken out while printing
35	ZBR_ERROR_CHECK_RIBBON	Incorrect ribbon installed
40	ZBR_ERROR_INVALID_MAGNETIC_DATA	Invalid magnetic encoding data
41	ZBR_ERROR_MAG_ENCODER_WRITE	Error while encoding a magnetic stripe
42	ZBR_ERROR_READING_ERROR	Error while reading a magnetic stripe
43	ZBR_ERROR_MAG_ENCODER_MECHANICAL	Magnetic encoder mechanical error
44	ZBR_ERROR_MAG_ENCODER_NOT_ RESPONDING	Magnetic encoder not responding
45	ZBR_ERROR_MAG_ENCODER_MISSING_OR_CARD_ JAM	Magnetic encoder is missing or the card is jammed before reaching the encoder
47	ZBR_ERROR_ROTATION_ERROR	Error while trying to flip the card
48	ZBR_ERROR_COVER_OPEN	Feeder Cover Lid is open (P110 and P120 only)
49	ZBR_ERROR_ENCODING_ERROR	Error while trying to encode on a magnetic stripe
50	ZBR_ERROR_MAGNETIC_ERROR	Magnetic encoder error
51	ZBR_ERROR_BLANK_TRACK	One or more of the tracks of the magnetic stripe are blank
52	ZBR_ERROR_FLASH_ERROR	Flash memory error
53	ZBR_ERROR_NO_ACCESS	Cannot access the printer
54	ZBR_ERROR_SEQUENCE_ERROR	Reception timeout, protocol errors
55	ZBR_ERROR_PROX_ERROR	Reception timeout, protocol errors
56	ZBR_ERROR_CONTACT_DATA_ERROR	Parameter error
57	ZBR_ERROR_PROX_DATA_ERROR	Parameter error

General SDK Error Codes

CODE	ERROR	POSSIBLE CAUSE
60	ZBR_SDK_ERROR_PRINTER_NOT_SUPPORTED	Printer not supported
61	ZBR_SDK_ERROR_CANNOT_GET_PRINTER_HANDLE	Unable to open handle to Zebra printer driver
62	ZBR_SDK_ERROR_CANNOT_GET_PRINTER_DRIVER	Cannot open printer driver
63	ZBR_SDK_ERROR_INVALID_PARAMETER	One of the arguments is invalid
64	ZBR_SDK_ERROR_PRINTER_BUSY	Printer is in use
65	ZBR_SDK_ERROR_INVALID_PRINTER_HANDLE	Invalid printer handle
66	ZBR_SDK_ERROR_CLOSE_HANDLE_ERROR	Error closing printer driver handle
67	ZBR_SDK_ERROR_COMMUNICATION_ERROR	Command failed due to communication error
68	ZBR_SDK_ERROR_BUFFER_OVERFLOW	Response too large for buffer
69	ZBR_SDK_ERROR_READ_DATA_ERROR	Error reading data
70	ZBR_SDK_ERROR_WRITE_DATA_ERROR	Error writing data
71	ZBR_SDK_ERROR_LOAD_LIBRARY_ERROR	Error loading SDK
72	ZBR_SDK_ERROR_INVALID_STRUCT_ALIGNMENT	Invalid structure alignment
73	ZBR_SDK_ERROR_GETTING_DEVICE_CONTEXT	Unable to create the device context for the driver
74	ZBR_SDK_ERROR_SPOOLER_ERROR	Print spooler error
75	ZBR_SDK_ERROR_OUT_OF_MEMORY	Operating system is out of memory
76	ZBR_SDK_ERROR_OUT_OF_DISK_SPACE	Operating system is out of disk space
77	ZBR_SDK_ERROR_USER_ABORT	Print job aborted by the user
78	ZBR_SDK_ERROR_APPLICATION_ABORT	Application aborted
79	ZBR_SDK_ERROR_CREATE_FILE_ERROR	Error creating file
80	ZBR_SDK_ERROR_WRITE_FILE_ERROR	Error writing file
81	ZBR_SDK_ERROR_READ_FILE_ERROR	Error reading file
82	ZBR_SDK_ERROR_INVALID_MEDIA	Invalid media

CODE	ERROR	POSSIBLE CAUSE
83	ZBR_SDK_ERROR_MEMORY_ALLOCATION	Insufficient system resources to perform necessary memory allocation
255	ZBR_SDK_ERROR_UNKNOWN_ERROR	An unknown but serious exception has occurred

Graphic Error Codes

CODE	ERROR	POSSIBLE CAUSE
8001	ZBR_GDI_ERROR_GENERIC_ERROR	Window API error, call GetLastError() function from Win32 API for error information
8002	ZBR_GDI_ERROR_INVALID_PARAMETER	One of the arguments is invalid
8003	ZBR_GDI_ERROR_OUT_OF_MEMORY	Operating system is out of memory
8004	ZBR_GDI_ERROR_OBJECT_BUSY	One of the objects specified in the API call is in use
8005	ZBR_GDI_ERROR_INSUFFICIENT_BUFFER	A buffer specified as an argument in the API call is not large enough
8006	ZBR_GDI_ERROR_NOT_IMPLEMENTED	Method is not implemented
8007	ZBR_GDI_ERROR_WIN32_ERROR	Method generated a Win32 error, call GetLastError() function from Win32 API for error information
8008	ZBR_GDI_ERROR_WRONG_STATE	Object called by the API is in an invalid state
8009	ZBR_GDI_ERROR_ABORTED	Method aborted
8010	ZBR_GDI_ERROR_FILE_NOT_FOUND	File not found
8011	ZBR_GDI_ERROR_VALUE_OVERFLOW	Arithmetic operation in the method caused a numeric overflow
8012	ZBR_GDI_ERROR_ACCESS_DENIED	Access denied to the specified file
8013	ZBR_GDI_ERROR_UNKNOWN_IMAGE_FORMAT	Specified image file format is unknown
8014	ZBR_GDI_ERROR_FONT_FAMILY_NOT_FOUND	Specified font is not installed
8015	ZBR_GDI_ERROR_FONT_STYLE_NOT_FOUND	Invalid font style
8016	ZBR_GDI_ERROR_NOT_TRUE_TYPE_FONT	Specified font is not a True Type font and cannot be used with GDI+
8017	ZBR_GDI_ERROR_UNSUPPORTED_GDIPLUS_ VERSION	Installed GDI+ version
8018	ZBR_GDI_ERROR_GDIPLUS_NOT_INITIALIZED	The GDI+ API is not initialized
8019	ZBR_GDI_ERROR_PROPERTY_NOT_FOUND	Specified property does not exist in the image
8020	ZBR_GDI_ERROR_PROPERTY_NOT_SUPPORTED	Specified property is not supported by the image format
8021	ZBR_GDI_ERROR_GRAPHICS_ALREADY_INITIALIZED	Graphic buffer has already been initialized
8022	ZBR_GDI_ERROR_NO_GRAPHIC_DATA	No data in the graphic buffer to print

CODE	ERROR	POSSIBLE CAUSE
8023	ZBR_GDI_ERROR_GRAPHICS_NOT_INITIALIZED	Graphics buffer has not been initialized
8024	ZBR_GDI_ERROR_GETTING_DEVICE_CONTEXT	Unable to create the device context for the driver
8025	ZBR_DLG_ERROR_DLG_CANCELED	User closed or canceled the DLG window
8026	ZBR_DLG_ERROR_SETUP_FAILURE	PrintDlg function failed to load the required resources
8027	ZBR_DLG_ERROR_PARSE_FAILURE	PrintDlg function failed to parse the strings in the [devices] section of the WIN.INI file
8028	ZBR_DLG_ERROR_RET_DEFAULT_FAILURE	PD_RETURNDEFAULT flag was specified in the Flags member of the PRINTDLG structure, but the hDevMode or hDevNames member was not NULL
8029	ZBR_DLG_ERROR_LOAD_DRV_FAILURE	PrintDlg function failed to load the device driver for the specified printer
8030	ZBR_DLG_ERROR_GET_DEVMODE_FAIL	Printer driver failed to initialize a DEVMODE structure
8031	ZBR_DLG_ERROR_INIT_FAILURE	PrintDlg function failed during initialization, and there is no more specific extended error code to describe the failure
8032	ZBR_DLG_ERROR_NO_DEVICES	No printer drivers were found
8033	8032 ZBR_DLG_ERROR_NO_DEFAULT_PRINTER	A default printer does not exist
8034	ZBR_DLG_ERROR_DN_DM_MISMATCH	Data in the DEVMODE and DEVNAMES structures describes two different printers
8035	ZBR_DLG_ERROR_CREATE_IC_FAILURE	PrintDlg function failed when it attempted to create an information context
8036	ZBR_DLG_ERROR_PRINTER_NOT_FOUND	The [devices] section of the WIN.INI file did not contain an entry for the requested printer
8037	ZBR_DLG_ERROR_DEFAULT_DIFFERENT	Error occurs when you store the DEVNAMES structure, and the user changes the default printer by using the Control Panel

Ethernet SmartCard Encoding SDK Error Codes

CODE	ERROR
65001	Device not open
65002	Device already open
65003	Device not available
65008	Invalid argument
65012	Buffer overflow
65021	Memory allocation error
65022	No devices found
65023	Disconnect error
65035	Unexpected error

Appendix B

Data Types

Card Types

ZBR_SYNCHRONOUS	=	1
ZBR_ISO_78163	=	2

Operating Modes

ZBR_ISO_MODE	=	0
ZBR_EMV_MODE	=	1

Printer Type

ZXP Series 3 Single-Sided Printing	=	31
ZXP Series 3 Dual-Sided Printing	=	32

Appendix C

Enumerations

C.1 Data Structures

```
public enum Overlay_Type
{
        FULL_OVERLAY = 0,
        NO_OVERLAY = 1,
        SELECTED_AREA = 2,
        SELECTED_BLANK = 3,
        BITMAP_BASED = 4,
        SMART_ISO = 5,
        SMART_AFNOR = 6,
        MAG_STRIPE = 7
}
```

Appendix D

Magnetic Encoders

With the magnetic stripe card encoder option, users can encode 3-track High-Coercivity (HiCo) or Low-Coercivity (LoCo) magnetic striped cards.

This appendix contains information detailing magnetic stripe encoding.

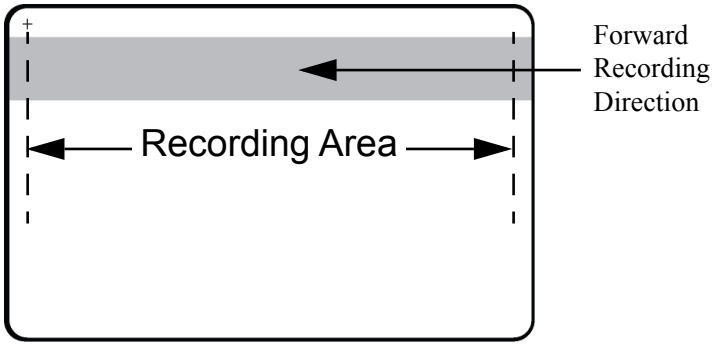
Magnetic Encoders

All printers with encoders write and read ANSI 4.16 and ISO 7811/2/3. Encoder track positions are fixed and cannot be modified.

Two encoder read-write head mounting positions exist:

- Below the Card Path -- The standard mounting that supports down-facing magnetic stripes when loading cards.
- Above the Card Path -- An optional mounting that supports up-facing magnetic stripes when loading cards.

The read-write heads are positioned just beyond the print head for both options.



Encoder Operation

The encoder executes commands received one at a time. When the encoder receives a command, it performs the requested action and reports the result. The printer cannot execute a new encoder command prior to completion of the previous encoder command.

Detailed encoder (and general printer) status information is reported to the host via an optional serial interface port only.

Write

The encoder, in default configuration, can write in the forward or reverse directions and then automatically perform a write-verifying data read. The printer then repositions the card to the print-ready position.

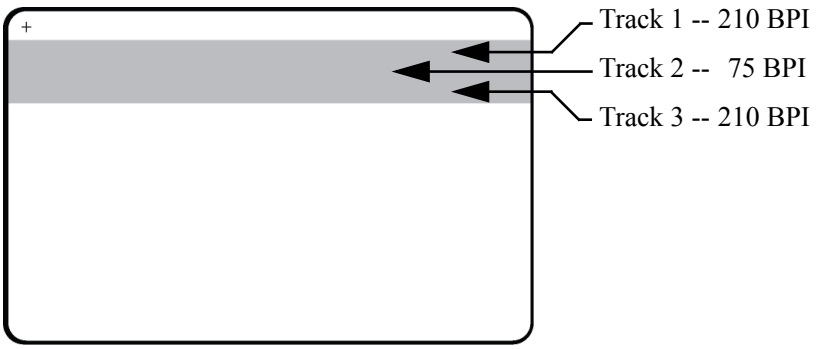
Note that for ISO encoding, the encoder attaches the start, stop, and LRC characters, which should not be included in data downloads.

Read

The encoder can only read (back to the host) a single track of data at a time.

Encoder Default Configuration

The encoder reads and writes standard ANSI/ISO track data formats in standard ANSI/ISO track locations. The following shows the three standard ANSI/ISO tracks.



Each track can be encoded and decoded with ASCII characters in the standard default ANSI/ISO data formats:

Track	Density	Data Format	Data Characters	Data Separator	Number of Characters
1	210 BPI	7 Bit (6 data, 1 parity)	Space $ ( ) - / Enter 0 through 9 A through Z (all caps)	^	79
2	75 BPI	5 Bit (4 data, 1 parity)	0 through 9	=	40
3	210 BPI	5 Bit (4 data, 1 parity)	1 through 9	=	107

The magnetic encoder can read or encode up to 3 tracks of digital information onto CR-80 cards incorporating a HiCo or LoCo magnetic stripe in the ANSI/ISO 7811 format.

Encoding for the three tracks uses the ISO 7811 format.

- Track 1 uses 210 BPI (bits per inch) encoding in the International Air Transport Association (IATA) format of 79 alphanumeric characters, at 7 bits per character.
- Track 2 uses 75 BPI encoding to store 40 numeric characters at 5 bits per character in American Banking Association (ABA) format.
- Track 3 uses 210 BPI encoding of 107 numeric characters at 5 bits per character in THRIFT format.

The ANSI/ISO data formats include a preamble (all zeros), a start character, data (7-bit or 5-bit as specified by ANSI/ISO), a stop character, and a longitudinal redundancy check (LRC) character. The 7-bit data format has 6 bits of encoded data and a parity bit. The 5-bit data format has 4 bits of encoded data and a parity bit.

The ANSI/ISO data formats include a data field separator (or delimiter) that allows parsing of the encoded track data. An example of separate data fields would be the ABA data format (Track 2) that includes a Primary Account Number (PAN) field and an account information field (for expiration date, country code, etc.).

Note that a user-specific custom format can also be employed.

Appendix E

Bar Codes

Bar codes vary in capacity, size, character sets, and density. Several industries have adopted specific coding and bar code formats. A selected bar code must match a code supported by the scanning equipment. All the bar codes offered by the card printers have the data characters, two quiet zones, and start and stop characters. The bar codes can include text as part of the printed bar code. Some of the bar codes include a printer-generated check digit (or data check sum) character automatically or as an option.

A command error condition occurs when image data extends beyond the addressable range of the image buffer. The bar code and text fields must remain within the addressable area of the image buffer. Each of the bar codes listed in this appendix have a formula to determine a bar code length.

Selecting a larger bar code width multiplier and a higher ratio of the narrow to wide bars (and spaces, where applicable) improves the general readability of a bar code. Also, wider bars and spaces increase the depth of field for improved performance with moving-beam lasers and other non-contact scanning devices.

This appendix contains a listing and explanation of the bar code types supported by Zebra card printers:

Code 39 (Code 3 of 9)

Code 39 encodes alphanumeric characters using five bars and four spaces. Of the nine, three are wide. The Ratio (R) determines wide-to-narrow bar and space widths. The minimum for a narrow bar or space is three dots or 0.010 inch (0.254 mm).

Supported Ratios of narrow-bar to wide-bar widths are 2:1, 5:2 (2.5:1), and 3:1.

The set of Characters (44) for Code 39 are as follows:

	Hexadecimal - Most Significant Digit								
Hexadecimal - Least Significant Digit	-	0	1	2	3	4	5	6	7
0	0	16	SP 32	0 48	64	P 80	96	112
1	1	17	33	1 49	A 65	Q 81	97	113
2	2	18	34	2 50	B 62	R 82	98	114
3	3	9	35	3 51	C 63	S 83	99	115
4	4	20	$ 36	4 52	D 64	T 84	100	116
5	5	21	% 37	5 53	E 69	U 85	101	117
6	6	22	38	6 54	F 70	V 86	102	118
7	7	23	39	7 55	G 71	W 87	103	119
8	8	24	40	8 56	H 72	X 88	104	120
9	9	25	41	9 57	I 73	Y 89	106	121
A	10	26	* 42	58	J 74	Z 90	107	122
B	11	27	+ 43	59	K 75	91	108	123
C	12	28	44	60	L 76	92	109	124
D	13	29	- 45	61	M 77	93	110	125
E	14	30	. 46	62	N 78	94	111	126
F	15	31	/ 47	63	O 79	95	112	127

To calculate the full length of a Code 39 bar code:

$$L = [(C+2)(3R + 7) - 1] X$$

Where L = Length of bar code
C = Number of characters
R = Ratio of wide-to-narrow bars
X = Number of dots times 0.0033 inches per dot (0.08847 mm per dot); for the 5:2 ratio, X = Dots times 2

The specified minimum recommended height is 0.25 inches (6.35 mm) or 75 dots. The recommend “Quiet Zone” is 0.25 inches (6.35mm or 75 dots) or, when larger, 10 times X.

Interleaved 2 of 5 (Code I 2/5)

The name Interleaved 2 of 5 derives from the method used to encode two characters. The bar code symbol pairs two characters, using bars to represent the first character and the interleaved spaces to represent the second character. Therefore, each character has two definitions, one for bars and the other for spaces. Each consists of two wide elements and three narrow elements. Bars and spaces are wide or narrow and the wide bars are set by the Ratio (R).

Interleaved 2 of 5 bar code supports numeric characters 0 through 9.

The printer automatically adds a leading zero (0) character to Code I 2/5 bar codes with an odd number of bar code data characters.

The supported ratio of narrow bar to wide bar widths are 2:1, 2:5 (2.5:1), and 3:1.

To calculate the full length of an Interleaved 2/5 bar code:

$$L = [C (2R + 3) + 6 + R] X$$

Where: L = Length of bar code
C = Number of characters
R = Ratio of wide-to-narrow bars (For 5:2, R=2.5)
X = Number of dots times 0.0033 inches per dot (0.08847 mm per dot)

The recommended bar code height is 0.25 inches (6.35 mm) or 75 dots. Ideally, the bar code height should be 15% of the bar code length. The recommend “Quiet Zone” is 0.25" (6.35mm or 75 dots) or, when larger, 10 times X.

Industrial 2 of 5 (Code 2/5)

Industrial 2 of 5 bar code is a low-density numeric bar code that does not require a checksum. It is a non-interleaved bar code that is easier to print than the Interleaved 2 of 5 bar code because check digits are not required. The Industrial 2 of 5 bar code symbology encodes all information in the width of the bars. Spaces carry no information. Bars are wide or narrow and the wide bars are set by the Ratio (R). Spaces are the same width as the narrow bars.

Industrial 2 of 5 bar code supports numeric characters 0 through 9.

The supported ratio of narrow bar to wide bar widths are 2:1, 5:2 (2.5:1), and 3:1.

To calculate the full length of a Industrial 2 of 5 bar code:

$$L = [\ C\ (2R + 8) + 14\ ]\ X$$

Where L = Length of bar code
C = Number of characters
R = Ratio of wide-to-narrow bars (For 5:2, R = 2.5)
X = Number of dots times 0.0033 inches per dot (0.08847 mm per dot); for the 5:2 ratio, X = Dots times 2

The minimum recommended bar code height is 0.25 inches (6.35 mm) or 75 dots. The recommend “Quiet Zone” is 0.25 inches (6.35mm or 75 dots) or, when larger, 10 times X.

EAN-8

European Article Numbering, now also called IAN (International Article Numbering), is the international standard bar code for retail food packages, corresponding to the Universal Product Code (UPC) in the United States. The symbology encodes a seven-digit EAN-8 number. The printer automatically generates an eighth Check Digit.

Numerous international agencies assign EAN code numbers and check digits.

EAN-8 Code supports numeric characters 0 through 9.

The printer ignores the ratio command parameter (narrow-bar to wide-bar width).

The equation to calculate the EAN-8 bar code length is:

$$L = (67)\ X$$

Where L = Length of bar code
X = Number of dots times 0.0033 inches per dot (0.08847 mm per dot)

EAN-8 bar code height, by specification, is six (6) individual EAN-8 bar code characters high. The following equation can be used to calculate the industry-specified height in dots:

$$H = (42)\ X$$

Where H = Height of bar code in dots
X = Bar code multiplier

Multiply the height of the bar code in dots by 0.0033 inches per dot (0.08847 mm per dot) to get the actual bar code height.

EAN-13

EAN-13 is one of two versions of the European Article Numbering (EAN) system and is a super set of UPC. EAN-13 has the same number of bars as UPC-A (Universal Product Code, version A) but encodes a 13th digit. The 12th and 13th digits define the country code. The codes 00-04 and 06-09 are assigned to the United States.

Numerous international agencies assign the EAN-13 code numbers.

EAN-13 Code supports numeric characters 0 through 9.

The printer ignores the ratio command parameter (narrow-bar to wide-bar width).

The equation to calculate the EAN-13 bar code length is:

$$L = (98)\ X$$

Where L = Length of bar code
X = Number of dots times 0.0033 inches per dot (0.08847 mm per dot)

EAN-13 bar code height, by specification, is six individual EAN-13 bar code characters high. The following equation can be used to calculate the industry-specified height in dots:

$$H = (42)\ X$$

Where H = Height of bar code in dots
X = Bar code multiplier

Multiply the height of the bar code in dots by 0.0033 inches per dot (0.08847 mm per dot) to get the actual bar code height.

UPC-A

UPC-A (Universal Product Code, version A) is the basic version of UPC and is usually the version seen on grocery store items in the United States. The symbology encodes10-digit UPC numbers. An 11th digit, at the beginning, indicates the type of product, and a 12th digit is a module check digit.

UPC-A code supports numeric characters 0 through 9.

The printer ignores the ratio command parameter (narrow-bar to wide-bar width).

The equation to calculate the UPC-A bar code length is:

$$L = (91)\ X$$

Where L = Length of bar code
X = Number of dots times 0.0033 inches per dot (0.08847 mm per dot)

UPC-A bar code height, by specification, is six individual UPC-A bar code characters high. The following equation can be used to calculate the industry-specified height in dots:

$$H = (42)\ X$$

Where H = Height of bar code in dots
X = Bar code multiplier

Multiply the height of the bar code in dots by 0.0033 inches per dot (0.08847 mm per dot) to get the actual bar code height.

Code 128, Subsets B & C

Code 128 is a high-density alpha-numeric bar code, which consists of a leading quiet zone, one of three start codes, the data itself, a check character, a stop character, and a trailing quiet zone. The Code 128 specification defines three “character sets” or “character modes” as Code 128 A, Code 128 B, and Code 128 C. Zebra printers support Code 128 B and Code 128 C.

Zebra printers, in Code 128 B mode, encode single-digital alpha-numerics as single bar code characters. Zebra printers, in Code 128 C mode, encode two numeric digits as a single bar code character.

VALUE	CODE A	CODE B	CODE C
0	SP	SP	0
1	!	!	1
2	"	"	2
3	#	#	3
4	$	$	4
5	%	%	5
6	&	&	6
7	'	'	7
8	(	(	8
9	)	)	9
10	*	*	10
11	+	+	11
12	,	,	12
13	-	-	13
14	.	.	14
15	/	/	15
16	0	0	16
17	1	1	17
18	2	2	18
19	3	3	19
20	4	4	20
21	5	5	21
22	6	6	22
23	7	7	23
24	8	8	24
25	9	9	25
26	:	:	26
27	;	;	27
28	<	<	28
29	=	=	29
30	>	>	30
31	?	?	31
32	@	@	32
33	A	A	33
34	B	B	34
35	C	C	35
36	D	D	36

VALUE	CODE A	CODE B	CODE C
37	E	E	37
38	F	F	38
39	G	G	39
40	H	H	40
41	I	I	41
42	J	J	42
43	K	K	43
44	L	L	44
45	M	M	45
46	N	N	46
47	O	O	47
48	P	P	48
49	Q	Q	49
50	R	R	50
51	S	S	51
52	T	T	52
53	U	U	53
54	V	V	54
55	W	W	55
56	X	X	56
57	Y	Y	57
58	Z	Z	58
59	[	[	59
60	\	\	60
61	]	]	61
62	^	^	62
63	_	_	63
64	NUL	`	64
65	SOH	a	65
66	STX	b	66
67	ETX	c	67
68	EOT	d	68
69	ENQ	e	69
70	ACK	f	70
71	BEL	g	71
72	BS	h	72
73	HT	I	73

VALUE	CODE A	CODE B	CODE C
74	LF	j	74
75	VT	k	75
76	FF	l	76
77	CR	m	77
78	SO	n	78
79	SI	o	79
80	DLE	p	80
81	DC1	q	81
82	DC2	r	82
83	DC3	s	83
84	DC4	t	84
85	NAK	u	85
86	SYN	v	86
87	ETB	w	87
88	CAN	x	88
89	EM	y	89
90	SUB	z	90
91	ESC	{	91
92	FS	|	92
93	GS	}	93
94	RS	~	94
95	US	DEL	95
96	FNC3	FNC3	96
97	FNC2	FNC2	97
98	SHIFT	SHIFT	98
99	Code C	Code C	99
100	Code B	FNC4	Code B
101	FNC4	Code A	Code A
102	FNC1	FNC1	FNC1
103	Start A	Start A	Start A
104	Start B	Start B	Start B
105	Start C	Start C	Start C

The printer ignores the ratio command parameter (narrow-bar to wide-bar width).

The equation to calculate the Code 128 B bar code length is:

$$L = [C (11) + 24] X$$

Where L = Length of bar code
C = Number of characters & checksum character
X = Number of dots times 0.0033 inches per dot (0.08847 mm per dot)

The equation to calculate the Code 128 C bar code length is:

$$L = [(11 C) / 2) + 24] X$$

Where L = Length of bar code
C = Number of characters (rounded up to the next even digit) & checksum character
X = Number of dots times 0.0033 inches per dot (0.08847 mm per dot)

The minimum recommended bar code height is 0.25 inches (6.35 mm) or 75 dots. Ideally the bar code height should be 15% of the bar code length. The recommend “Quiet Zone” is 0.25 inches (6.35mm or 75 dots) or, when larger, 10 times X.

Appendix F

Worldwide Support

For Technical Support or Repair Services, contact the appropriate facility listed below.

North America and Latin America - Technical Support

T: +1 877 ASK ZEBRA (877 275 9327)
+1 847 913 2259
E: ts1@zebra.com

North America and Latin America - Repair Services

Before returning any equipment to Zebra Technologies Corporation for in-warranty or out-of-warranty repair, contact Repair Services for a Repair Order (RO) number. Mark the RO number clearly on the outside of the box. Ship the equipment, freight prepaid, to the address listed below:

Zebra Technologies Repair Services
333 Corporate Woods Parkway
Vernon Hills, IL 60061

webform: www.zebra.com/repair
T: +1 877 ASK ZEBRA (877 275 9327)
E: repair@zebra.com

Europe, Middle East, and Africa - Technical Support

Language		Phone	Email
Arabic		+971 (0)46058220	zebraCareUAE@zebra.com
Dutch		+31 (0)33 450 50 48	ZebraCareBNL@zebra.com
English	(UK)	+44 (0)1628 556 225	zebraCareUK@zebra.com
	(Sweden)	+46 (0)8 594 709 88	zebraCareUK@zebra.com
	(South Africa)	+27 (0)11 201 7712 / 0860 393272	zebracareSA@zebra.com
French		+33 (0) 1 53 48 12 74	zebraCareFR@zebra.com
German		+49 (0) 2159 676 870	zebraCareDE@zebra.com
Hebrew		+97 236 498 140	ZebraCareIL@zebra.com
Italian		+39 0 2 575 06388	ZebraCareIT@zebra.com
Polish		+48 223 801 980	zebraCarePL@zebra.com
Russian		+7 495 739 5993	ZebraCareRU@zebra.com
Spanish		+34 (0) 917 992 896	zebraCareES@zebra.com
Turkish		+90 212 314 1010	zebraCareTR@zebra.com

For further assistance, contact:

Zebra Technologies Card Printer Solutions
Dukes Meadow
Millboard Road, Bourne End
Buckinghamshire SL8 5XF, UK

T: +44 (0) 1628 556 025
F: +44 (0) 1628 556 001
E: cardts@zebra.com

Europe, Middle East, and Africa - Repair Services

Before returning any equipment to Zebra Technologies Corporation for in-warranty or out-of-warranty repair, contact your supplier for a Return Materials Authorization (RMA) number, or contact one ot the following repair centers for support and instructions:

Type of repair and location	Phone	Email
Depot Repair in Germany, Austria, Switzerland	+49 (0) 2159 676 870	zebracareDE@zebra.com
Depot Repair in France	+33 (0) 1 53 48 12 74	zebracareFR@zebra.com
Depot and On-Site Repair in UK and Ireland	+44 (0) 1628 556 225	zebracareUK@zebra.com
Depot Repair in South Africa	+27 (0) 11 201 7777	-
Depot Repair in Middle East	+971 (0) 46058220	support_dxb@emitac.ae

For further assistance, contact:

For assistance anywhere in the EMEA, contact After Sales Customer Services at:

T: + 44 (0) 177 2 69 3069
E: ukrma@zebra.com

Asia Pacific - Technical Support

Zebra Technologies Asia Pacific Pte. Ltd.
120 Robinson Road
#06-01 Parakou Building
Singapore 068913

T: +65 6858 0722
F: +65 6885 0838
E: tsasiapacific@zebra.com

Asia Pacific - Repair Services

Before returning any equipment to Zebra Technologies Corporation for in-warranty or out-of warranty repair, contact Repair Services for a Return Materials Authorization (RMA) number. Repack the equipment in the original packing material, and mark the RMA number clearly on the outside. Ship the equipment, freight prepaid, to either address listed below:

Zebra Technologies Asia Pacific Pte. Ltd.
No.5 Changi North Way Level 3
Singapore 498771
Agility Building

T: +65 6546 2670 ext 3203 and 3204
F: +65 6546 5328
E: APACRepair@zebra.com

Zebra Website

http://www.zebra.com

km.zebra.com (Knowledge Base)