

# ZMotif™ SDK



**ZEBRA**

## Software Developer Reference Manual

ZEBRA and the stylized Zebra head are trademarks of Zebra Technologies Corporation, registered in many jurisdictions worldwide. All other trademarks are the property of their respective owners.  
© 2019 Zebra Technologies Corporation and/or its affiliates. All rights reserved.

Information in this document is subject to change without notice. The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of those agreements.

For further information regarding legal and proprietary statements, please go to:

SOFTWARE: <http://www.zebra.com/linkoslegal>

COPYRIGHTS: <http://www.zebra.com/copyright>

WARRANTY: <http://www.zebra.com/warranty>

END USER LICENSE AGREEMENT: <http://www.zebra.com/eula>

## Terms of Use

### Proprietary Statement

This manual contains proprietary information of Zebra Technologies Corporation and its subsidiaries ("Zebra Technologies"). It is intended solely for the information and use of parties operating and maintaining the equipment described herein. Such proprietary information may not be used, reproduced, or disclosed to any other parties for any other purpose without the express, written permission of Zebra Technologies.

### Product Improvements

Continuous improvement of products is a policy of Zebra Technologies. All specifications and designs are subject to change without notice.

### Liability Disclaimer

Zebra Technologies takes steps to ensure that its published Engineering specifications and manuals are correct; however, errors do occur. Zebra Technologies reserves the right to correct any such errors and disclaims liability resulting therefrom.

### Limitation of Liability

In no event shall Zebra Technologies or anyone else involved in the creation, production, or delivery of the accompanying product (including hardware and software) be liable for any damages whatsoever (including, without limitation, consequential damages including loss of business profits, business interruption, or loss of business information) arising out of the use of, the results of use of, or inability to use such product, even if Zebra Technologies has been advised of the possibility of such damages. Some jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation or exclusion may not apply to you.

# Contents

<b>1 • Introduction .....</b>	<b>1</b>
About this Manual .....	1
Required Skills .....	1
Zebra Card Printers.....	2
ZMotif Overview Introduction .....	2
ZMotif Architecture .....	2
The ZMJ layer .....	3
The ZMC layer .....	3
The Windows print job sequence .....	4
Multiple hosts in a network.....	4
 <b>2 • Software Development Kit Plus (SDK) .....</b>	 <b>5</b>
Introduction .....	5
SDK Diagram .....	5
 <b>3 • Graphics COM Object .....</b>	 <b>7</b>
Properties.....	7
Methods .....	8
GetSDKProductVersion .....	9
GetSDKVersion .....	10
InitGraphics .....	11
CloseGraphics .....	12
ClearGraphics.....	13
DrawImage .....	14
DrawEllipse.....	15
DrawFillEllipse .....	16
DrawLine .....	17
DrawRectangle .....	18
DrawFillRectangle .....	19
DrawFillRoundedRectangle .....	20

DrawRoundedRectangle .....	21
DrawTextRect .....	22
DrawTextString .....	23
CreateBitmap .....	24
ExtractBlackData .....	25
SetBlackExtractionLevels .....	26
ApplyColorProfile .....	27
AdjustBrightness .....	28
AdjustColorScale .....	29
AdjustContrast .....	30
AdjustSaturation .....	31
AdjustGamma .....	32
ConvertToGrayScale .....	33
RotateHue .....	34
CropImage .....	35
SharpenImage .....	36
ImageFileToByteArray .....	37
ImageFromByteArray .....	38
IPictureFromByteArray .....	39
IntegerFromColor .....	40
IntegerFromColorName .....	41
LoadImageProfile .....	42
SaveImageProfile .....	43
ExtractHalfPanelData .....	44
<b>4 • Barcode .....</b>	<b>45</b>
Properties .....	45
Methods .....	48
ClearBarcode .....	49
DrawBarcode (3 overloaded methods) .....	50
DrawBarcodeToStream .....	51
<b>5 • Job .....</b>	<b>53</b>
Properties .....	53
Methods .....	54
GetDriverName .....	55
Open .....	56
Close .....	57
ClearGraphicsLayers .....	58
BuildGraphicsLayers .....	59

BuildGraphicsLayersEx .....	60
PrintGraphicsLayers .....	61
LoadJobConfigFile.....	62
TestPrint .....	63
PrintGraphicsLayersWithMagData .....	64
MagDataOnly .....	66
ReadMagData .....	67
SmartCardDataOnly .....	69
GetJobList .....	71
GetJobStatus .....	72
GetJobStatusEx.....	74
JobAbort .....	76
JobCancel.....	77
JobReprint .....	78
JobResume .....	79
JobRetry .....	80
SmartCardRetry.....	81
ClearError .....	82
Reset .....	83
GetPrinters .....	84
GetBroadcastConfiguration .....	85
SetBroadcastConfiguration.....	86
ReserveDevice .....	87
GetReservationStatus .....	88
IsDeviceInSession .....	89
ReleaseDevice .....	90
GetSDKVersion .....	91
GetSDKProductVersion .....	92
EjectCard .....	93
RejectCard.....	94
ReturnCardToFeeder .....	95
PositionCard .....	96
ReadBarcodeData .....	97
 <b>6 • Job Control .....</b>	 <b>99</b>
Properties.....	99
Methods .....	101
JobConfiguration .....	102
SideConfiguration .....	103
MagConfiguration .....	104
SmartCardConfiguration .....	105

ConfigureDataEncryption .....	106
GetDataEncryptionConfiguration .....	107
GetAvailableCardTypes .....	108
GetCardTypeInfoInformation .....	109
IntegerFromColorName .....	110
SetKPanelOptimization .....	111
KPanelOverwrite .....	112
SetMonoConvType .....	113
<b>7 • Device .....</b>	<b>115</b>
Properties .....	115
Methods .....	117
GetPrinterStatus .....	118
ResumeFromStandby .....	119
GetCapabilities .....	120
GetConfiguration .....	121
SetConfiguration .....	122
GetCustomCardConfiguration .....	123
SetCustomCardConfiguration .....	124
GetAvailableMemory .....	125
GetDeviceInfo .....	126
GetFilmParams .....	127
GetLog .....	128
GetMagneticEncoderConfiguration .....	129
GetMagStartSentinelOffset .....	130
SetMagStartSentinelOffset .....	131
GetNetworkParams .....	132
SetNetworkParams .....	133
GetPanelPowerLevels .....	134
GetPanelPowerLevelsEx .....	135
SetPanelPowerLevels .....	136
SetPanelPowerLevelsEx .....	137
GetRejectCardCount .....	138
ClearRejectBinCount .....	139
GetRibbonParams .....	140
GetSensorStates .....	141
GetSensorValues .....	142
GetSmartCardConfiguration .....	143
GetStatusMessageString .....	144
GetTotalCardCount .....	145
GetTransferTempOffsets .....	146

SetTransferTempOffsets .....	147
GetDisplayedOCPMessage .....	148
BuildOCPMessage .....	149
ClearOCPMessage .....	150
DisplayOCPMessage .....	151
SelectOCPButton .....	152
GetCleaningIntervals .....	153
SetCleaningIntervals .....	154
UpgradeFirmware .....	155
LockPrinter .....	156
UnlockPrinter .....	157
GetNVMEMData .....	158
SetNVMEMData .....	159
GetWirelessStatus .....	160
GetWirelessRadioStatus .....	161
GetWirelessParams .....	162
SetWirelessParams .....	164
ScanWirelessAccessPoints .....	165
GetCalibrationTableData .....	166
GetCalibrationTableNames .....	167
GetClock .....	168
GetI2CErrorStats .....	169
LoadCalibrationTableData .....	170
SetClock .....	171
<b>8 • Custom Mag Settings .....</b>	<b>173</b>
Properties .....	173
Methods .....	174
GetMagTrackBitsPerChar .....	175
GetMagTrackDataParity .....	176
GetMagTrackDensity .....	177
GetMagTrackSentinelFormat .....	178
SetMagTrackBitsPerChar .....	179
SetMagTrackDataParity .....	180
SetMagTrackDensity .....	181
SetMagTrackSentinelFormat .....	182
<b>9 • Laminator .....</b>	<b>183</b>
Methods .....	183
CalibrateLaminate .....	184

RestoreDefaultConfiguration .....	185
SaveParameters .....	186
GetLaminateParams .....	187
GetLaminatorStatus .....	188
GetBottomLaminateOffsets .....	189
SetBottomLaminateOffsets .....	190
GetTopLaminateOffsets .....	191
SetTopLaminateOffsets .....	192
GetLaminationSpeedOffsets .....	193
SetLaminationSpeedOffsets .....	194
GetLaminatorOffsets .....	195
SetLaminatorOffsets .....	196
GetLaminatorSensorStates .....	197
GetLaminatorSensorValues .....	198
GetLaminatorCardCount .....	199
GetLaminatorInfo .....	200
LoadParameters .....	201
GetOdometerValues .....	202
 <b>10 • Utilities .....</b>	 <b>203</b>
Methods .....	203
ByteArrayToVariantArray .....	204
BytePtrToVariantArray .....	205
IntArrayToVariantArray .....	206
LongArrayToVariantArray .....	207
VariantArrayToByteArray .....	208
VariantArrayToIntArray .....	209
VariantArrayToLongArray .....	210
 <b>Appendix A • Error Codes .....</b>	 <b>211</b>
 <b>Appendix B • SDK Enumerations .....</b>	 <b>221</b>
 <b>Appendix C • XML Documents .....</b>	 <b>229</b>
 <b>Appendix D • Sample Code .....</b>	 <b>245</b>
 <b>Appendix E • Page Size &amp; Orientation Conventions .....</b>	 <b>255</b>

# Introduction

## About this Manual

This manual contains information for software developers intending to write applications for Zebra ZMotif compatible card printers.

The ZMotif Application Programming Interface (ZAPI) and Software Development Kit (ZMotif SDK) provides functions to access Zebra ZMotif-based card printers and features.

ZAPI runs on the following Windows Operating Systems:

- Windows 7 (32- and 64-bit)
- Windows Server 2008 (32- and 64-bit)
- Windows 8 and 8.1 (32- and 64-bit)
- Windows 10 (32- and 64-bit)

## Required Skills

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

## ZMotif Overview Introduction

The ZMotif SDK facilitates the communication of data between a host computer and a Zebra color printer and to provide a solid framework to accommodate future development needs. It is a print job comprehending language that includes job control commands as well as commands to transfer data configure the printer and return information from the printer.

The ZMotif SDK is intended for use with "modern-architecture" printers that support multi-tasking with sufficient memory to buffer entire print jobs.

The ZMotif SDK is based on an extensible underlying printer communication and job control mechanism. This low-level device-specific protocol separates out printer communication data flow, and the actual print job-specific data.

# ZMotif SDK Architecture

ZMotif comprises two distinct independent layers:

1. ZMJ a ZMotif Job description protocol that sits above ZMC.
2. ZMC ZMotif real-time Communications protocol that handles commands to and responses from the target printer.

Most ZMC commands are concerned with housekeeping operations such as configuration control status reporting and error reporting. In fact only one of many ZMC commands Write Data calls on ZMJ to deliver specs and data for the job to be printed. The separation of “job” and “communications” portions of the language is important in three ways:

1. It meshes nicely with the printing architectures of popular operating systems;
2. It simplifies transactions over the communications interface;
3. It facilitates debugging.

## The ZMJ layer

This layer contains all the commands a printer needs to complete the desired job which might be printing one or more single-sided or double-sided cards or receiving data such as graphics calibration tables fonts and firmware upgrades.

ZMJ is a small set of commands small because ZMJ itself serves mostly as a means of conveying information to the printer in the form of XML documents. For example a print job would begin with the ZMJ command named “Begin Job” consisting mostly of a “Job Control” XML document specifying how the card is to be printed and encoded.

The Job Control XML might be complete in itself containing all the data necessary to execute the job. Alternatively especially if the job includes graphics it will be augmented by one or more appended ZMJ “Send Graphic” commands each referencing a specific tag in the Job Control XML.

Unlike those of ZMC commands in ZMJ are not necessarily executed in real time. Jobs in ZMJ can be stored on disk or buffered internally within the printer and de-spooled at a later time for processing.

## The ZMC layer

This layer sits on top of any bi-directional streaming communications channel such as USB or Ethernet. ZMC manages real-time communications tasks such as sending data and receiving error messages. One of ZMC’s tasks is transferring a ZMJ print job from the host to the printer (but note that to ZMC a print job is simply a data stream it doesn’t interact with in any way).

In addition to sending data and receiving error messages ZMC allows the printer to respond to requests for its status including media availability and any controllable parameters thus allowing the host to configure job specifications in the most appropriate way.

## The Windows print job sequence

Assuming that the host is running a Zebra-provided Windows printer driver a typical sequence is the following:

1. The Windows application composes the document then sends it to the Windows printer driver.
2. The driver converts the document’s description to a ZMJ print job package comprising at least two commands Begin Job and End Job. The driver also adds to the Job Control XML a ZMJ Job Identifier (a UUID or in a Windows environment a GUID).

Important: Specifications and data for the print job are contained in the Begin Job command the payload of which is an XML document known as the **Job Control XML**. The package may also contain within the Begin Job-End Job scope secondary ZMJ commands to convey information not included in the XML such as graphics.

3. The driver sends the print job package to the language monitor which buffers all the data between ZMJ Start Job and ZMJ End Job commands.
4. When notified by the printer that it is ready to receive the job the language monitor issues a ZMC Start Action command then delivers the job in one or more ZMC Write Data commands followed by a ZMC End Action command. This signifies that the job transfer has been completed at which time the printer should begin processing.

Note: On receiving a Start Action command from the language monitor the printer returns an Action ID number. This can be used together with the ZMJ Job Identifier and the Spooler Job ID number to uniquely identify the print job.

## Multiple hosts in a network

In an Ethernet network where the printer may be shared by multiple hosts, ZMotif SDK commands allow the printer to receive data from only one host at a time.

Additionally ZMotif SDK commands require that the printer must have received ALL of a print job from a given host before beginning to receive any part of the next print job either from that host or any other host.



# Software Development Kit (SDK)

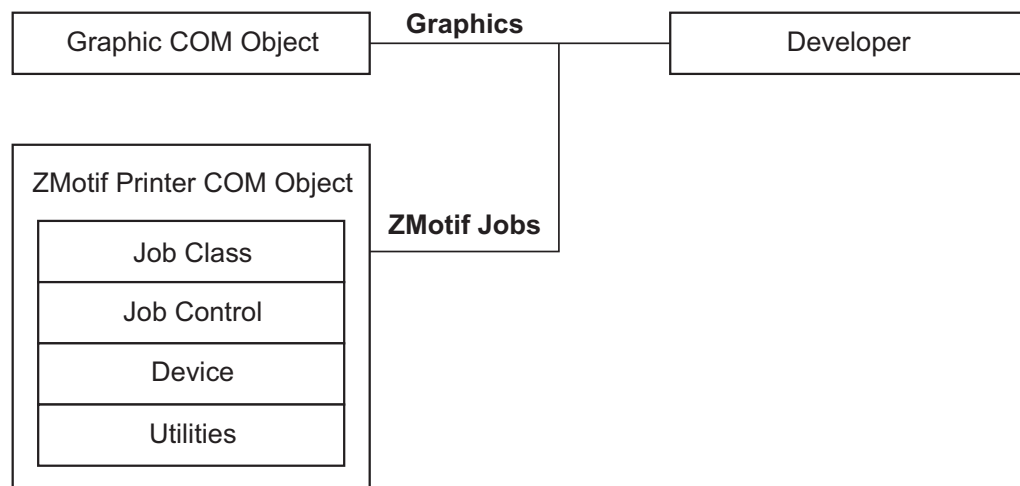
## Introduction

The ZMotif Software Development Kit (SDK) provides developers a platform for designing applications for ZMotif compatible devices. It contains two registered COM objects: Graphics and ZMotif Printer.

The Graphics Object assists in the creation of ID Card bitmap images. It contains all of the basic methods necessary for creating ID Cards. The ZMotif Printer Object contains all of the methods necessary for creating ZMotif jobs and commands.

The ZMotif SDK does not require a printer driver to communicate with a ZMotif-supported device and it supports both USB and Ethernet connections.

## SDK Diagram





# Graphics COM Object

## Properties

<u>Integer</u>	
TextContrast	gets/sets the contrast value for text
<u>Interface</u>	
Barcode	returns the Barcode interface
<u>Enumeration</u>	
ImageRotation	gets/sets image rotation used with DrawImage routines
MonochromeConversion	gets/sets the type of monochrome conversion to use
PrinterModel	gets/sets the printer model in use
TextRendering	gets/sets the text rendering methodology to use
<u>String</u>	
ColorProfile	gets/sets the color profile

## Methods

GetSDKProductVersion . . . . .	9
GetSDKVersion . . . . .	10
InitGraphics . . . . .	11
CloseGraphics . . . . .	12
ClearGraphics . . . . .	13
DrawImage . . . . .	14
DrawEllipse . . . . .	15
DrawFillEllipse . . . . .	16
DrawLine . . . . .	17
DrawRectangle . . . . .	18
DrawFillRectangle . . . . .	19
DrawFillRoundedRectangle . . . . .	20
DrawRoundedRectangle . . . . .	21
DrawTextRect . . . . .	22
DrawTextString . . . . .	23
CreateBitmap . . . . .	24
ExtractBlackData . . . . .	25
SetBlackExtractionLevels . . . . .	26
ApplyColorProfile . . . . .	27
AdjustBrightness . . . . .	28
AdjustColorScale . . . . .	29
AdjustContrast . . . . .	30
AdjustSaturation . . . . .	31
AdjustGamma . . . . .	32
ConvertToGrayScale . . . . .	33
RotateHue . . . . .	34
CropImage . . . . .	35
SharpenImage . . . . .	36
ImageFileToByteArray . . . . .	37
ImageFromByteArray . . . . .	38
IPictureFromByteArray . . . . .	39
IntegerFromColor . . . . .	40
IntegerFromColorName . . . . .	41
LoadImageProfile . . . . .	42
SaveImageProfile . . . . .	43
ExtractHalfPanelData . . . . .	44

## GetSDKProductVersion

Description: Retrieves the product version of the SDK.

Syntax:       void GetSDKProductVersion( out string ProductVersion )

Parameters:   ProductVersion           [out]product version of SDK

Returns:       nothing

## GetSDKVersion

Description: Retrieves the version of the SDK.

Syntax:       void GetSDKVersion(  
                                    out byte               major  
                                    out byte               minor  
                                    out byte               build  
                                    out byte               revision )

Parameters:   major               [out]major version number of SDK  
              minor               [out]minor version number of SDK  
              build               [out]build version number of SDK  
              revision            [out]revision version number of SDK

Returns:       nothing

## InitGraphics

**Description:** Creates a Windows device context and initializes a graphic buffer for storing graphic layers.

**Syntax 1:**

```
void InitGraphics(
    int                maxWidth
    int                maxHeight
    ImageOrientationEnum imgOrientation
    RibbonTypeEnum      ribType )
```

**Parameters:**

maxWidth	[in ]maximum graphic buffer width
maxHeight	[in ]maximum graphic buffer height
imgOrientation	[in ]specifies if card image is portrait or landscape
ribType	[in ]specifies a ribbon type

**Returns:** nothing

**Syntax 2:**

```
void InitGraphics(
    int                maxWidth
    int                maxHeight
    ImageOrientationEnum imgOrientation
    RibbonTypeEnum      ribType
    int                fillColor )
```

**Parameters:**

maxWidth	[in ]maximum graphic buffer width
maxHeight	[in ]maximumgraphic buffer height
imgOrientation	[in ]specifies if the card image is portrait or landscape
ribType	[in ]specifies a ribbon type
fillColor	[in ]identifies if there is a fill color (RGB) e.g. white = 0xFFFFFF no fill color = -1

**Returns:** nothing

**Note:** Setting maxWidth and maxHeight to 00 defaults the image size to match ID card dimensions 1024 x 648.

### CloseGraphics

**Description:** Releases the device context value the graphic buffer memory and the mutex.

**Syntax:**        void CloseGraphics( void )

**Parameters:** nothing

**Returns:**        nothing

**Note:**         If the graphics SDK is instantiated CloseGraphics must be called as part of clean up or the mutex will not be released.

## ClearGraphics

Description: Clears the graphic buffer.

Syntax:       void ClearGraphics( void )

Parameters: nothing

Returns:       nothing

## DrawImage

Description: Places an image in the graphic buffer.

**Syntax 1:**

```
void DrawImage(
    ref byte[]          imageData
    ImagePositionEnum   position
    int                 imgWidth
    int                 imgHeight
    float               opacity )
```

Parameters:

imageData	[in ]image byte array
position	[in ]image position
imgWidth	[in ]image width in dots (300 dpi)
imgHeight	[in ]image height in dots (300 dpi)
opacity	[in ]image opacity level (0 to 100)

Returns: nothing

**Syntax 2:**

```
void DrawImage(
    ref byte[]          imageData
    float               x
    float               y
    int                 imgWidth
    int                 imgHeight
    float               opacity )
```

Parameters:

imageData	[in ]image byte array
x	[in ]x coordinate top left corner of the image
y	[in ]y coordinate top left corner of the image
imgWidth	[in ]image width in dots (300 dpi)
imgHeight	[in ]image height in dots (300 dpi)
opacity	[in ]image opacity level (0 to 100)

Returns: nothing

**Syntax 3:**

```
void DrawImage(
    ref byte[]          imageData
    float               x
    float               y
    int                 imgWidth
    int                 imgHeight
    float               opacity
    int                 transparencyColorLow
    int                 transparencyColorHigh )
```

Parameters:

imageData	[in ]image byte array
x	[in ]x coordinate top left corner of the image
y	[in ]y coordinate top left corner of the image
imgWidth	[in ]image width in dots (300 dpi)
imgHeight	[in ]image height in dots (300 dpi)
opacity	[in ]image opacity level (0 to 100)
transparencyColorLow	[in ]low color value starting transparent color
transparencyColorHigh	[in ]high color value ending transparent color

Returns: nothing

Note: All colors between transparencyColorLow and transparencyColorHigh will be transparent.

Use *ImageFileToByteArray(filename)* to get imageData from a file.

## DrawEllipse

Description: Draws an ellipse in the graphic buffer.

Syntax:       void DrawEllipse(                   float                   x  
  float                   y  
  float                   width  
  float                   height  
  float                   thickness  
  int                     color )

Parameters: x                   [in ]x position of top-left corner of rectangle  
             y                   [in ]y position of top-left corner of rectangle  
             width               [in ]width of the ellipse in dots (300 dpi)  
             height              [in ]height of the ellipse in dots (300 dpi)  
             thickness           [in ]line thickness for the ellipse in dots  
             color               [in ]RGB color value

Returns:       nothing

## DrawFillEllipse

Description: Draws a solid ellipse in the graphic buffer.

Syntax:       void DrawFillEllipse(                               float                       x  
   float                       y  
   float                       width  
   float                       height  
   int                          fillColor )

Parameters:   x                               [in ]x position of top-left corner of rectangle  
              y                               [in ]y position of top-left corner of rectangle  
              width                           [in ]width of the ellipse in dots (300 dpi)  
              height                          [in ]height of the ellipse in dots (300 dpi)  
              fillColor                      [in ]color of ellipse RGB color value

Returns:       nothing

## DrawLine

Description: Draws a line in the graphic buffer.

Syntax:       void DrawLine(                               float               x1  
   float               y1  
   float               x2  
   float               y2  
   int                 color  
   float               thickness )

Parameters:   x1                               [in ]starting x position for the line  
              y1                               [in ]starting y position for the line  
              x2                               [in ]ending x position for the line  
              y2                               [in ]ending y position for the line  
              color                           [in ]RGB color value  
              thickness                      [in ]line thickness

Returns:       nothing

## DrawRectangle

Description: Draws a rectangle in the graphic buffer.

Syntax:       void DrawRectangle(                   float                   x  
  float                   y  
  float                   width  
  float                   height  
  float                   thickness  
  int                     color )

Parameters: x                   [in ]x position of top-left corner of rectangle  
             y                   [in ]y position of top-left corner of rectangle  
             width               [in ]rectangle width in dots (300 dpi)  
             height              [in ]rectangle height in dots (300 dpi)  
             thickness           [in ]line thickness  
             color               [in ]RGB color value

Returns:       nothing

## DrawFillRectangle

Description: Draws a solid rectangle in the graphic buffer.

Syntax:       void DrawFillRectangle(  
  float                x  
  float                y  
  float                width  
  float                height  
  int                   fillColor )

Parameters:  x                   [in ]x position of top-left corner of rectangle  
              y                   [in ]y position of top-left corner of rectangle  
              width               [in ]rectangle width in dots (300 dpi)  
              height              [in ]rectangle height in dots (300 dpi)  
              fillColor           [in ]color of rectangle RGB color value

Returns:     nothing

## DrawFillRoundedRectangle

Description: Draws a solid rectangle with rounded corners in the graphic buffer.

Syntax: `void DrawFillRoundedRectangle(  
float x  
float y  
float width  
float height  
float radius  
int fillColor )`

Parameters: x	[in ]x position of top-left corner of rectangle
y	[in ]y position of top-left corner of rectangle
width	[in ]rectangle width in dots (300 dpi)
height	[in ]rectangle height in dots (300 dpi)
radius	[in ]radius of the semi-circle formed by the rounded corners
fillColor	[in ]color of rectangle RGB color value

Returns: nothing

Sample:

```
float x = 10;
float y = 10;
float width = 10;
float height = 10;
float radius = 20;
int fillColor = RGB(100 100 100);

graphics.DrawFillRoundedRectangle(x y width height radius
fillColor );
```

## DrawRoundedRectangle

Description: Draws a rectangle with rounded corners in the graphic buffer.

Syntax: `void DrawRoundedRectangle(`

```

float x
float y
float width
float height
float radius
float thickness
int color )
```

Parameters:

<code>x</code>	[in ]x position of top-left corner of rectangle
<code>y</code>	[in ]y position of top-left corner of rectangle
<code>width</code>	[in ]rectangle width in dots (300 dpi)
<code>height</code>	[in ]rectangle height in dots (300 dpi)
<code>thickness</code>	[in ]rectangle thickness in dots (300 dpi)
<code>color</code>	[in ]color of rectangle RGB color value

Returns: nothing

Sample:

```

float x = 10;
float y = 10;
float width = 10;
float height = 10;
float thickness = 20;
int color = RGB(100 100 100);

graphics.DrawRoundedRectangle(x y width height thickness color);
```

## DrawTextRect

Description: Draws text in the graphic buffer within the specified rectangle.

**Syntax 1:**

```
void DrawTextRect(
    float          x
    float          y
    float          rectWidth
    float          rectHeight
    TextAlignmentEnum alignment
    string         text
    string         font
    float          fontSize
    FontTypeEnum   fontStyle
    int            color )
```

Parameters:

x	[in ]x position of top-left corner of rectangle
y	[in ]y position of top-left corner of rectangle
rectWidth	[in ]rectangle width in dots (300 dpi)
rectHeight	[in ]rectangle height in dots (300 dpi)
alignment	[in ]alignment within the rectangle
text	[in ]text data
font	[in ]font name
fontSize	[in ]font point size
fontStyle	[in ]font style
color	[in ]RGB value

Returns: nothing

**Syntax 2:**

```
void DrawTextRect(
    float          x
    float          y
    float          rectWidth
    float          rectHeight
    float          angle
    TextAlignmentEnum alignment
    string         text
    string         font
    float          fontSize
    FontTypeEnum   fontStyle
    int            color )
```

Parameters:

x	[in ]x position of top-left corner of rectangle
y	[in ]y position of top-left corner of rectangle
rectWidth	[in ]rectangle width in dots (300 dpi)
rectHeight	[in ]rectangle height in dots (300 dpi)
angle	[in ]text drawing angle
alignment	[in ]alignment within the rectangle
text	[in ]text data
font	[in ]font name
fontSize	[in ]font point size
fontStyle	[in ]font style
color	[in ]RGB value

Returns: nothing

## DrawTextString

Description: Draws text in the graphic buffer.

**Syntax 1:**      void DrawTextString(

float	x
float	y
string	text
string	font
float	fontSize
FontTypeEnum	fontStyle
int	color )

Parameters:    x                    [in ]x position of top-left corner of text  
                   y                    [in ]y position of top-left corner of text  
                   text                [in ]text data  
                   font                [in ]font name  
                   fontSize           [in ]font point size  
                   fontStyle          [in ]font style  
                   color               [in ]RGB value

Returns:        nothing

**Syntax 2:**      void DrawTextString(

float	x
float	y
float	angle
TextAlignmentEnum	alignment
string	text
string	font
float	fontSize
FontTypeEnum	fontStyle
int	color )

Parameters:    x                    [in ]x position of top-left corner of text  
                   y                    [in ]y position of top-left corner of text  
                   angle                [in ]text drawing angle  
                   alignment          [in ]text alignment  
                   text                [in ]text data  
                   font                [in ]font name  
                   fontSize           [in ]font point size  
                   fontStyle          [in ]font style  
                   color               [in ]RGB value

Returns:        nothing

### CreateBitmap

Description: Creates a bitmap image from graphic buffer.

Syntax: `byte[] CreateBitmap( out int dataLen )`

Parameters: `dataLen` [out]image size in bytes

Returns: Bitmap image byte array

## ExtractBlackData

Description: Performs black data extraction on a given image.

Syntax:       void ExtractBlackData(  
  ref byte[]                imageData  
  ref byte[]                colorImg  
  ref byte[]                blackImg )

Parameters:   imageData               [in ]given bitmap image's data  
              colorImg                [out]given image's color data  
              blackImg                [out]given image's black data

Returns:       nothing

Note:         If no black data is found in the given image the blackImg parameter will be null.

## SetBlackExtractionLevels

**Description:** Provides a small amount of manipulation of the threshold which determines whether or not a specific pixel contains black data.

**Syntax:**

```
void SetBlackExtractionLevels(  
                                int             redLevel  
                                int             greenLevel  
                                int             blueLevel )
```

**Parameters:**

redlevel	[in ]red color threshold
greenLevel	[in ]green color threshold
blueLevel	[in ]blue color threshold

**Returns:** nothing

**Sample:**

```
int redLevel = 25;  
int greenLevel = 25;  
int blueLevel = 25;  
graphics.SetBlackExtractionLevels(redLevel greenLevel blueLevel);
```

## ApplyColorProfile

Description: Applies a color profile to an existing bitmap and returns a new bitmap created from the original bitmap and the applied profile.

```
Syntax:      byte[] ApplyColorProfile(
                                string          profilePath
                                byte[]         origImage )
```

Parameters:	profilePath	[in] color profile to apply including path to file's location.
	origImage	[in] byte array representing original image.

Returns: Byte array representing the new image created from the application of the color profile and original image.

Note: A color profile cannot be applied to a non-color image.

## AdjustBrightness

Description: Adjusts image brightness in the graphic buffer.

Syntax:       void AdjustBrightness( float brightnessLevel )

Parameters: brightnessLevel       [in ]percentage value {-100 to +100}

Returns:       nothing

## AdjustColorScale

Description: Adjusts the RGB color scales in the graphic buffer.

Syntax:       void AdjustColorScale(                               float                       redLevel  
   float                       greenLevel  
   float                       blueLevel )

Parameters:   redLevel                       [in ]red contrast level  
                 greenLevel                   [in ]green contrast level  
                 blueLevel                   [in ]blue contrast level

Returns:       nothing

Note:          Values are from -100 to +100.

### AdjustContrast

Description: Adjusts image contrast for all colors in the graphic buffer.

Syntax:       void AdjustContrast( float contrastLevel )

Parameters: contrastLevel               [in ]contrast level

Returns:       nothing

Note:         Values are from -100 to +100.

## AdjustSaturation

Description: Adjusts image saturation in the graphic buffer.

Syntax:       void AdjustSaturation( float saturationLevel )

Parameters: saturationLevel           [in ]image saturation level

Returns:       nothing

Note:         Values are from -100 to +100.

## AdjustGamma

Description: Sets the gamma level for the image.

Syntax:       void AdjustGamma( int gammaLevel )

Parameters: gammaLevel                   [in ]new gamma level to be applied

Return:       nothing

Sample:       int gammaLevel = 100;  
              graphics.AdjustGamma(gammaLevel);

## ConvertToGrayScale

**Description:** Converts the color image in the graphic buffer to monochrome applying the same scale factor to each color.

**Syntax:** void ConvertToGrayScale( float scaleFactor )

**Parameters:** scaleFactor [in ]scale factor; 0.1 to 1.0

**Returns:** nothing

**Description:** Converts the color image in the graphic buffer to monochrome applying specific red green and blue scale factors.

**Syntax:** void ConvertToGrayScale( float red float green float blue )

**Parameters:** red [in ]red scale factor; 0.1 to 1.0  
 green [in ]green scale factor; 0.1 to 1.0  
 blue [in ]blue scale factor; 0.1 to 1.0

**Returns:** nothing

**Description:** Creates a monochrome image from a color image.

**Syntax:** void ConvertToGrayScale( byte[] imageData ref byte[] grData )

**Parameters:** imageData [in ]color image data to be converted  
 grData [out]monochrome image data created from color image

**Returns:** nothing

## RotateHue

Description: Rotate image color spectrum.

Syntax:       void RotateHue( float rotation )

Parameters: rotation                   [in ]degrees of rotation; 0 to 360

Returns:       nothing

## CropImage

Description: Creates a cropped image from the supplied image.

Syntax:       void CropImage(                               ref byte[]                   imageData  
  int                               x  
  int                               y  
  int                               width  
  int                               height  
  ref byte[]                   croppedImg)

Parameters:   imageData                               [in ]image data to be cropped  
                  x                                   [in ]x-coordinate of upper left corner of  
  the cropped image  
                  y                                   [in ]y-coordinate of upper left corner of  
  the cropped image  
                  width                               [in ]width of cropped image in dots (300 dpi)  
                  height                             [in ]height of cropped image in dots (300 dpi)  
                  croppedImg                         [out]cropped image data

Returns:       nothing

## SharpenImage

Description: Sharpens an image in the byte array.

Syntax:       void SharpenImage(                               ref byte[]                       imageData  
  int                               level )

Parameters:   imageData                               [in ]array containing the image to sharpen  
              level                                   [in ]sharpening level; 0 to 100

Returns:       nothing

## ImageFileToByteArray

Description Creates an image byte array from file.

Syntax: `byte[] ImageFileToByteArray( string filename )`

Parameters: filename [in ]name of the file that contains the image

Returns: Byte array of image created from image file.

## ImageFromByteArray

Description: Creates a bitmap from an image byte array.

Syntax:       Bitmap ImageFromByteArray( ref byte[] imageData )

Parameters:   imageDataByte               [in ]byte array containing the image data

Returns:       Bitmap image created from image byte array.

## IPictureFromByteArray

Description: Returns IPicture data from an image byte array.

Syntax:        object IPictureFromByteArray( ref byte[] imageData )

Parameters:   imageDatabyte                [in ]byte array containing the image data

Returns:        IPicture data from image byte array returned as an object.

## IntegerFromColor

Description: Gets the integer value for a selected system color.

Syntax:       int IntegerFromColor( Color color )

Parameters: color                               [in ]system color

Returns:       Integer value which represents the system color object.

## IntegerFromColorName

Description: Gets the integer value for a color.

Syntax:        `int IntegerFromColorName( string colorName )`

Parameters:   `colorName`                      `[in ]color name`

Returns:       `Integer value which represents the system color name.`

## LoadImageProfile

Description: Loads an image profile from a file.

Syntax:       void LoadImageProfile( string filename )

Parameters: filename                   [in ]profile file name and path

Returns:       nothing

## SaveImageProfile

Descripton: Saves an image profile to a file.

Syntax: void SaveImageProfile( string filename )

Parameters: filename [in ]profile file name and path

Returns: nothing

## ExtractHalfPanelData

**Description:** Performs data extraction on a given image when using a half-panel ribbon.

**Syntax:**

```
void ExtractHalfPanelData(
    PrinterModelTypeEnum    model,
    ref byte[]               sourceImage,
    ref byte[]               colorRegion,
    ref byte[]               nonColorRegion)
```

**Parameters:**

Model	[in ]printer model to receive the extracted data; see Appendix B for enumeration values
sourceImage	[out]bitmap image to perform extraction
colorRegion	[out]bitmap containing the color data
nonColorRegion	[out]bitmap containing the non-color data

**Return:** Nothing

# Barcode

## Properties

<u>Boolean</u>	
AddChecksum	gets/sets a flag which determines if achecksum must be generated and attached to the value to encode
AntiAlias	gets/sets a flag which determines if AntiAlias effect must be applied to all the texts in the barcode image
AutoSize	gets/sets a flag which determines if the barcode image is automatically resized to display its entire contents.
DisplayChecksum	gets/sets a flag which will display the barcode's checksum if set to true
DisplayCode	gets/sets a flag which determines if the value to encode will be displayed in the barcode image
DisplayStopStartChars	gets/sets a flag which determines if the start & stop characters must be displayed in the barcode image
HIBCFormattedHumanReadableText	gets/sets a flag which determines if the human readable text is formatted as specified by HIBC. (Zeros are displayed as Slashed-Zeros and space characters are displayed as underscores)
UseQuietZoneForText	gets/sets a flag which determines if the quiet zones are to be used for drawing text code and human readable text
<u>Integer</u>	
BackColor	gets/sets the background color
BarColor	gets/sets the color of the barcode bars
BorderColor	gets/sets the barcode control border color
ForeColor	gets/sets the text color when rendering the code and human readable text
TextForeColor	gets/sets the text color when rendering the DisplayText property

<u>Interface</u>	
AztecCodeProperties	returns the AztecCode symbology properties class
BarcodeMarginProperties	returns the BarcodeMargin interface
BearerBarProperties	returns the BearerBarProperties interface
CodabarProperties	returns the CodabarProperties interface
Code128Properties	returns the Code128Properties interface
Code16KProperties	returns the Code16KProperties interface
Code39And93Properties	returns the Code39 and 93Properties interface
DataMatrixProperties	returns the DataMatrix symbology properties interface
EanUpcProperties	returns the EAN/ UPC symbology properties interface
GS1DataBarAndRSSEExpandedStackedProperties	returns the GS1 DataBar Expanded Stacked and RSS Expanded Stacked properties interface
Isbt128Properties	returns the Isbt128 properties interface
MaxiCodeProperties	returns the MaxiCode properties interface
MicroQRProperties	returns the MicroQR properties interface
MSIProperties	returns the MSI properties interface
Pdf417Properties	returns the PDF417 symbology properties interface
PharmacodeProperties	returns the Pharmacode symbology properties interface
PlanetProperties	returns the Planet symbology properties interface
PostalStateProperties	returns the PostalState symbology properties interface
PostnetProperties	returns the Postnet symbology properties interface
QRCodeProperties	returns the QRCode symbology properties interface
TelepenProperties	returns the Telepen properties interface
USPSProperties	returns the USPS properties interface
<u>Double</u>	
BarcodeBottomMargin	gets/sets the margin below the barcode bars
BarcodeHeight	gets/sets the barcode image height (the AutoSize property must be set to false)
BarcodeTopMargin	gets/sets the height of the margin above the barcode bars
BarcodeWidth	gets/sets the barcode image width (the AutoSize property must be set to false)
BarHeight	gets/sets the height of the barcode bars
BarRatio	gets/sets the ratio of the difference between the width of the wide barcode bars and the width of the narrow barcode bars
BarWidth	gets/sets the width of the narrow barcode bars
BarWidthAdjustment	Sets/returns the width adjustment for the bars
BorderWidth	gets/sets the barcode image border width
QuietZoneWidth	gets/sets the width of the quiet zone

## Barcode

String

DisplayHumanReadableText

gets/sets the value to encode as human readable text

DisplayText

gets/sets additional text to display in the barcode image in addition to the value to encode

ValueToEncode

gets/sets the value to be encoded

Enumeration

BarcodeType

gets/sets Barcode type

CodeAlignment

gets/sets the text alignment for the DisplayCode property

Rotation

gets/sets the rotation angle to apply to the barcode

TextAlignment

gets/sets the text alignment of the barcode image

Object

Font

gets/sets the font to use for rendering the code and human readable text

TextFont

gets/sets the font to use for rendering the DisplayText property

## Methods

ClearBarcode .....	49
DrawBarcode (3 overloaded methods) .....	50
DrawBarcodeToStream .....	51

### ClearBarcode

Description: Resets all properties to their default values.

Syntax: `void ClearBarcode()`

Parameters: None

Returns: nothing

## DrawBarcode (3 overloaded methods)

**Description 1:** Draws the selected barcode symbology using barcode property values.

Syntax:        void DrawBarcode()

Parameters:   None

Returns:        nothing

**Description 2:** Draws the selected barcode symbology into a user defined rectangle.

Syntax:        void DrawBarcode(  
                                      float                                x  
                                      float                                y  
                                      float                                width  
                                      float                                height  
                                      float                                scale )

Parameters:   x                    [in ]x position of top-left corner of rectangle  
                  y                   [in ]y position of top-left corner of rectangle  
                  width               [in ]rectangle width in dots (300 dpi)  
                  height              [in ]rectangle height in dots (300 dpi)  
                  scale               [in ]scale value

Returns:        nothing

**Description 3:** Draws the selected barcode symbology into a user defined rectangle.

Syntax:        void DrawBarcode(  
                                      float                                x  
                                      float                                y  
                                      float                                width  
                                      float                                height  
                                      RotationTypeEnum               rotation  
                                      string                               font  
                                      int                                fontSize  
                                      FontTypeEnum                fontStyle  
                                      float                                scale )

Parameters:   x                    [in ]x position of top-left corner of rectangle  
                  y                   [in ]y position of top-left corner of rectangle  
                  width               [in ]rectangle width in dots (300 dpi)  
                  height              [in ]rectangle height in dots (300 dpi)  
                  rotation            [in ]rotation to apply to barcode (see  
    enumerations in [Appendix B](#))  
                  font                [in ]font type to use to render barcode  
                  fontStyle           [in ]style of font (see enumerations in  
    [Appendix B](#))  
                  scale               [in ]scale value

Returns:        nothing

## DrawBarcodeToStream

Description: Draws a barcode to the selected stream.

Syntax: `void DrawBarcodeToStream( ref Stream stream )`

Parameters: `stream` [in ] the stream to receive the barcode

Returns: `nothing`

Sample: 

```
System.IO.Stream stream = new System.IO.MemoryStream();  
graphics.Barcode.DrawBarcodeToStream(ref stream);
```



# Job

## Properties

### Boolean

EnableOCPreprint

gets/sets a flag indicating whether or not the printer's OCP will display a Reprint button

IsOpen

indicates if a connection to a device is active, read only

ReadResponse

sets/gets if a response is to be read after sending ZMC WriteData

### Class

FileSystem

returns a FileSystem class object

### Enumeration

EventLogLevel

gets/sets/EventLogTypeEnum - a value indicating the detail level of information to be logged

### Float

EthernetOpenTimeout

sets/gets the time out value for opening an Ethernet connection

### Interface

CustomMagSettings

returns the CustomMagSettings Interface

Device

returns the Device Interface

JobControl

returns the JobControl Interface

Laminator

returns the Laminator Interface

Utilities

returns the Utilities Interface

### Short

HostTCPPort

sets/gets host's TCP/IP port address

HttpPort

gets/sets the host's HTTP port value

## Methods

GetDriverName .....	55
Open .....	56
Close .....	57
ClearGraphicsLayers .....	58
BuildGraphicsLayers .....	59
BuildGraphicsLayersEx .....	60
PrintGraphicsLayers .....	61
LoadJobConfigFile .....	62
TestPrint .....	63
PrintGraphicsLayersWithMagData .....	64
MagDataOnly .....	66
ReadMagData .....	67
SmartCardDataOnly .....	69
GetJobList .....	71
GetJobStatus .....	72
GetJobStatusEx .....	74
JobAbort .....	76
JobCancel .....	77
JobReprint .....	78
JobResume .....	79
JobRetry .....	80
SmartCardRetry .....	81
ClearError .....	82
Reset .....	83
GetPrinters .....	84
GetBroadcastConfiguration .....	85
SetBroadcastConfiguration .....	86
ReserveDevice .....	87
GetReservationStatus .....	88
IsDeviceInSession .....	89
ReleaseDevice .....	90
GetSDKVersion .....	91
GetSDKProductVersion .....	92
EjectCard .....	93
RejectCard .....	94
ReturnCardToFeeder .....	95
PositionCard .....	96
ReadBarcodeData .....	97

## GetDriverName

Description: Retrieves the printer name from the printer driver.

Syntax:        `string GetDriverName( string deviceName )`

Parameters:   `deviceName`            [in ]serial number of ZXP Printer

Returns:       `printer name being used by printer driver`

Sample:        `string DriverName = job.GetDriverName(deviceName);`

## Open

Description: Establishes a connection to a ZMotif device.

Syntax: `short Open( string deviceName )`

Parameters: `deviceName` [in ]name of the ZMotif device

Returns: error code (see [Appendix A](#))

Sample:

```
//USB Connection:
Job job = new Job();
try
{
    string deviceName = "06C102100019"; //printer serial number
    short alarm = job.Open(deviceName);
}
catch (Exception ex)
{
    string errMsg = ex.Message;
}

//Ethernet Connection:
Job job = new Job();
try
{
    string deviceName = "10.1.5.123"; //printer IP address
    short alarm = job.Open(deviceName);
}
catch (Exception ex)
{
    string errMsg = ex.Message;
}

//Printer driver installed/USB connection:
Job job = new Job();
try
{
    //printer driver name
    string deviceName = "Zebra ZXP Series 8 USB Card Printer";
    short alarm = job.Open(deviceName);
}
catch (Exception ex)
{
    string errMsg = ex.Message;
}

//Printer driver installed/Ethernet connection:
Job job = new Job();
try
{
    //printer driver name
    string deviceName = "Zebra ZXP Series 8 Network Card Printer";
    short alarm = job.Open(deviceName);
}
catch (Exception ex)
{
    string errMsg = ex.Message;
}
}
```

## Close

Description: Closes the connection to a ZMotif Device.

Syntax: `void Close()`

Parameters: nothing

Returns: nothing

Sample:

```
try
{
    if (job.IsOpen)
        job.Close();
}
catch (Exception ex)
{
    string errMsg = ex.Message;
}
finally //be sure to release the interface to avoid memory leaks{
    do
    {
        Thread.Sleep(10);

    } while (Marshal.FinalReleaseComObject(job) > 0);
}
```

Note: As an alternative to calling it each time the Close method is called, the do-while loop can be called prior to the application shutting down to prevent memory leaks.

## ClearGraphicsLayers

Description: Erases all graphic layers.

Syntax: `void ClearGraphicsLayers()`

Parameters: nothing

Returns: nothing

Sample: `job.ClearGraphicsLayers();`

## BuildGraphicsLayers

Description: Builds a graphic image layer.

```
Syntax:      void BuildGraphicsLayers (
                                SideEnum          side,
                                PrintTypeEnum      printType,
                                int                xOffset,
                                int                yOffset,
                                int                imgOpacity,
                                int                fillColor,
                                GraphicTypeEnum    graphicType,
                                object             graphicData )
```

```
Parameters:  side           [in]specifies the graphic layer's card side
              printType      [in]type of print to perform (see Appendix B
                           for enumeration values)
              xOffset        [in]x offset
              yOffset        [in]y offset
              imgOpacity     [in]opacity value
              fillColor       [in]layer background fill color (RGB);
                           -1 indicates no fill color
              graphicType    [in]image format (see Appendix B for
                           enumeration values)
              graphicData    [in]image to import
```

Returns: nothing

Note: The first layer build will be the background, and the last layer will be the foreground.

Sample: See sample for PrintGraphicsLayers or PrintGraphicsLayersWithMagData

## BuildGraphicsLayersEx

**Description:** Builds a graphic image layer which includes controlling the ribbon panel movement direction, panel count for movement, and panel offset information. The layer can also be defined to overprint another image.

**Syntax:**

```
void BuildGraphicsLayersEx (
    SideEnum          side,
    PrintTypeEnum      printType,
    int               xOffset,
    int               yOffset,
    int               imgOpacity,
    int               fillColor,
    PanelDirectionEnum panelDirection,
    PanelCountEnum     panelCount,
    int               panelOffset,
    bool              overprint,
    GraphicTypeEnum    graphicType,
    object             graphicData )
```

**Parameters:**

side	[in]specifies the graphic layer's card side (see <a href="#">Appendix B</a> for enumeration values)
printType	[in]type of print to perform (see <a href="#">Appendix B</a> for enumeration values)
xOffset	[in]x offset
yOffset	[in]y offset
imgOpacity	[in]opacity value
fillColor	[in]layer background fill color (RGB); -1 indicates no fill color
panelDirection	[in]direction of ribbon panel movement: advance or reverse (see <a href="#">Appendix B</a> for enumeration values)
panelCount	[in]number of panels to move for panel direction (see <a href="#">Appendix B</a> for enumeration values)
panelOffset	[in]current panel
overprint	[in]determines if image should be printed over
graphicType	[in]image format (see <a href="#">Appendix B</a> for enumeration values)
graphicData	[in]image to import

**Returns:** nothing

**Note:** The first layer build will be the background, and the last layer will be the foreground.

**Sample:**

```
job.BuildGraphicsLayersEx(SideEnum.Front,
    PrintTypeEnum.Color, 0, 0, 1, -1,
    PanelDirectionEnum.AdvancePanel,
    PanelCountEnum.OnePanel, 10,
    false, GraphicTypeEnum.BMP,
    graphicData);
```

## PrintGraphicsLayers

Description: Prints all layers built by BuildGraphicsLayers.

```
Syntax:      void PrintGraphicsLayers(
                int                copies,
                out int             actionID)
```

Parameters:	copies	[in ] number of copies to print
	actionID	[out]returned by a ZMotif device identifying a job's ID

Returns: nothing

Note: If the card's source and destination locations are not assigned for the current print job, the default locations `FeederSourceEnum.CardFeeder`, and `DestinationTypeEnum.Eject` will be assigned automatically for the card's source and destination locations respectively if no previous print job has been created. If a previous print job has been created, its source and destination locations will be used for the current print job.

```

Sample:
// Prints front and back side images
Job job = new Job();
try
{
    byte[] bmpFront = ImageToByteArray("Front.bmp");
    byte[] bmpBack = ImageToByteArray("Back.bmp");
    job.Open(deviceName);
    job.BuildGraphicsLayers(SideEnum.Front, PrintTypeEnum.Color,
        0, 0, 0, GraphicTypeEnum.BMP, bmpFront);
    job.BuildGraphicsLayers(SideEnum.Back, PrintTypeEnum.MonoK,
        0, 0, 0, GraphicTypeEnum.BMP, bmpBack);
    int copies = 1;
    //assign the card's source and destination locations for
    //the print job:
    job.JobControl.FeederSource = FeederSourceEnum.CardFeeder;
    job.JobControl.Destination = DestinationTypeEnum.Eject;
    job.PrintGraphicsLayers(copies, out actionID);
    job.ClearGraphicsLayers();
    while(true)
    {
        job.GetJobStatus(actionID, out uuidJob,
            out printingStatus,
            out errorCode, out copiesCompleted,
            out copiesRequested, out magStatus,
            out contactStatus,
            out contactlessStatus);
        if (printingStatus == "done_ok" ||
            printingStatus == "cleaning_up" ||
            printingStatus == "done_error" ||
            printingStatus == "cancelled_by_user" ||
            printingStatus == "cancelled_by_error") break;
    }
    if error status
        jobs can be aborted or retried
}
catch (Exception ex)
{ errMsg = ex.Message;
}
finally
{
    job = null;
}
}

```

## LoadJobConfigFile

Description: Notifies the SDK to use the specified xml configuration document for the job.

Syntax: `LoadJobConfigFile( string configFilePath )`

Parameters: `configFilePath` [in ] document containing configuration information

Returns: `none`

Note: The complete path to the configuration file must be supplied each time the file is to be used.

Sample: `job.LoadJobConfig(configFilePath);`

## TestPrint

Description: Sends a test image to a device.

```
Syntax:      void TestPrint(
                byte      testImage,
                int        copies,
                bool        doubleSide,
                out int     actionID )
```

Parameters:	testImagetest	[in ] image number
	copies	[in ] number of tests
	doubleSide	[in ] identifies if test print is single-sided or dual-sided
	actionID	[out] returned by a ZMotif device identifying a job's ID

Returns: nothing

Note: If the card's source and destination locations are not assigned for the current print job, the default locations `FeederSourceEnum.CardFeeder`, and `DestinationTypeEnum.Eject` will be assigned automatically for the card's source and destination locations respectively if no previous print job has been created. If a previous print job has been created, its source and destination locations will be used for the current print job.

```

Sample:
//assign the card's source and destination locations for
//the print job:
job.JobControl.FeederSource = FeederSourceEnum.CardFeeder;
job.JobControl.Destination = DestinationTypeEnum.Eject;
{   job.TestPrint(1, 1, true, out actionID);
    while(true)
    {   alarm = job.GetJobStatus(actionID, out uuidJob,
        out printingStatus, out errorCode,
        out copiesCompleted, out copiesRequested,
        out magStatus, out contactStatus,
        out contactlessStatus);
        if (printingStatus == "done_ok" ||
            printingStatus == "cleaning_up" ||
            printingStatus == "done_error" ||
            printingStatus == "cancelled_by_user" ||
            printingStatus == "cancelled_by_error") break;
    }
    if error status
        jobs can be aborted or retried
}
catch (Exception ex)
{   errMsg = ex.Message;
}

```

## PrintGraphicsLayersWithMagData

Description: Encodes the magnetic data and prints the graphics layers.

Syntax:        void PrintGraphicsLayersWithMagData(  
     int                              copies,  
     string                      track1,  
     string                      track2,  
     string                      track3,  
     out int                          actionID)

Parameters:    copies                  [in ]number of copies to print and encode  
                  track1                 [in ]null or "" indicates no data to encode  
                  track2                 [in ]null or "" indicates no data to encode  
                  track3                 [in ]null or "" indicates no data to encode  
                  actionID              [out]returned by a ZMotif device identifying  
    a job's ID

Returns:        nothing

Note:            If the card's source and destination locations are not assigned for the current print job, the default locations FeederSourceEnum.CardFeeder, and DestinationTypeEnum.Eject will be assigned automatically for the card's source and destination locations respectively if no previous print job has been created. If a previous print job has been created, its source and destination locations will be used for the current print job.

Sample:         See next page

```

Sample: // Prints front and back side images
Job job = new Job();
try
{
    byte[] bmpFront = ImageToByteArray("Front.bmp");
    byte[] bmpBack = ImageToByteArray("Back.bmp");
    job.Open(deviceName);
    job.BuildGraphicsLayers(SideEnum.Front, PrintTypeEnum.Color,
        0, 0, 0, GraphicTypeEnum.BMP, bmpFront);
    job.BuildGraphicsLayers(SideEnum.Back, PrintTypeEnum.MonoK,
        0, 0, 0, GraphicTypeEnum.BMP, bmpBack);
    job.JobControl.MagConfiguration(SideEnum.Front,
        MagCoercivityEnum.HighCo, MagEncodingTypeEnum.ISO,
        MagDataFormatEnum.Ascii, true);
    int copies = 1;
    string track1Data = "ABCDEFGH";
    string track2Data = "12345678";
    string track3Data = "87654321";
    //assign the card's source and destination locations for
    //the print job:
    job.JobControl.FeederSource = FeederSourceEnum.CardFeeder;
    job.JobControl.Destination = DestinationTypeEnum.Eject;
    job.PrintGraphicsLayersWithMagData(copies, track1Data,
        track2Data, track3Data, out actionID);
    while(true)
    {
        job.GetJobStatus(actionID, out uuidJob,
            out printingStatus, out errorCode, out copiesCompleted,
            out copiesRequested, out magStatus,
            out contactStatus, out contactlessStatus);
        if (printingStatus == "done_ok" ||
            printingStatus == "cleaning_up" ||
            printingStatus == "done_error" ||
            printingStatus == "cancelled_by_user" ||
            printingStatus == "cancelled_by_error" ||
            magStatus.Contains("error")) break;
    }
    if error status
        jobs can be aborted or retried
}
catch (Exception ex)
{ string errMsg = ex.Message;
}
finally
{
    job = null;
}

```

## MagDataOnly

Description: Sends a magnetic encoding job, does not print.

Syntax:       void MagDataOnly(  
                                   int                                  copies,  
                                   string                              track1,  
                                   string                              track2,  
                                   string                              track3,  
                                   out int                              actionID)

Parameters:   copies                  [in ]number of cards to encode  
               track1                  [in ]null or "" indicates no data to encode  
               track2                  [in ]null or "" indicates no data to encode  
               track3                  [in ]null or "" indicates no data to encode  
               actionID                [out]returned by a ZMotif device identifying  
   a job's ID

Returns:       nothing

Note:           If the card's source and destination locations are not assigned for the current print job, the default locations FeederSourceEnum.CardFeeder, and DestinationTypeEnum.Eject will be assigned automatically for the card's source and destination locations respectively if no previous print job has been created. If a previous print job has been created, its source and destination locations will be used for the current print job.

Sample:        Job job = new Job();  
               try  
               {   job.Open(deviceName);  
                   //assign the card's source and destination locations for  
                   //the print job:  
                   job.JobControl.FeederSource = FeederSourceEnum.CardFeeder;  
                   job.JobControl.Destination = DestinationTypeEnum.Eject;  
                   job.MagDataOnly (1, "TRACK1DATA", "22222222", "33333333",  
                                     out actionID);  
                   while (true)  
                   {   alarm = job.GetJobStatus(actionID, out uuidJob,  
                       out printingStatus, out errorCode,  
                       out copiesCompleted, out copiesRequested,  
                       out magStatus, out contactStatus,  
                       out contactlessStatus);  
                       if (printingStatus == "done\_ok" ||  
                           printingStatus == "cleaning\_up" ||  
                           printingStatus == "done\_error" ||  
                           printingStatus == "cancelled\_by\_user" ||  
                           printingStatus == "cancelled\_by\_error" ||  
                           magStatus.Contains("error") break;  
                       }  
                       if error status  
                           jobs can be aborted or retried  
                   }  
                   catch (Exception ex)  
                   {   errMsg = ex.Message;  
                       }  
               }

## ReadMagData

Description: Reads one or more magnetic track.

```
Syntax:      void ReadMagData(
                DataSourceEnum    tracksToRead,
                out string         track1,
                out string         track2,
                out string         track3,
                out int            actionID);
```

Parameters:	tracksToRead	[in] number of track to read (see <a href="#">Appendix B</a> for enumeration values)
	track1	[out] Track 1 data
	track2	[out] Track 2 data
	track3	[out] Track 3 data
	actionID	[out] returned by a ZMotif device identifying a job's ID

Returns: nothing

Note: If the card's source and destination locations are not assigned for the current print job, the default locations `FeederSourceEnum.CardFeeder`, and `DestinationTypeEnum.Eject` will be assigned automatically for the card's source and destination locations respectively if no previous print job has been created. If a previous print job has been created, its source and destination locations will be used for the current print job.

```
Sample:      Job job = new Job();
             try
             {   job.Open(deviceName);
                 //assign the card's source and destination locations for
                 //the print job:
                 job.JobControl.FeederSource = FeederSourceEnum.CardFeeder;
                 job.JobControl.Destination = DestinationTypeEnum.Eject;
                 job.ReadMagData(DataSourceEnum.Track1Data |
                                DataSourceEnum.Track2Data |
                                DataSourceEnum.Track3Data,
                                out track1, out track2, out track3, out actionID);
             }
             catch (Exception ex)
             {   errMsg = ex.Message;
             }
             }
```

EIN Example: See next page

EIN Example: Here is the example code demonstrating how to work with EIN:

```
String Track1Data = "";
String Track3Data = "";
String EIN = "";
int actionID = 0;

//sets the EIN track number:
job.JobControl.DataSource = DataSourceEnum.Track2Data;

//read the EIN from track 2:
job.ReadMagData(DataSourceEnum.Track2Data, out Track1Data,
    out EIN, out Track3Data, out actionID);

//determine if EIN is correct:
if (EIN.Length > 0)
{
    //insert your logic here for making this determination
}
else //EIN not encoded on track 2
{
    //insert your logic here for processing empty EIN track
}
```

## SmartCardDataOnly

Description: Starts a smart card encode job, does not print.

Syntax:        void SmartCardDataOnly(  
    int                                copies,  
    out int                            actionID)

Parameters:    copies                    [in ] numner of cards to encode  
                  actionID                [out] returned by a ZMotif device identifying  
    a job's ID

Returns:        nothing

Note 1:        SmartCardDataOnly moves a card to the smart card encoding station and suspends the job; the job is either completed via the JobResume function or aborted via the JobAbort function.

Note 2:        If the card's source and destination locations are not assigned for the current print job, the default locations FeederSourceEnum.CardFeeder, and DestinationTypeEnum.Eject will be assigned automatically for the card's source and destination locations respectively if no previous print job has been created. If a previous print job has been created, its source and destination locations will be used for the current print job.

Sample:        See next page

```

Sample:  Job job = new Job();
        try
        {
            job.Open(deviceName);
            job.JobControl.SmartCardConfiguration(SideEnum.Front,
                SmartCardTypeEnum.Contact, true);
            //assign the card's source and destination locations for
            //the print job:
            job.JobControl.FeederSource = FeederSourceEnum.CardFeeder;
            job.JobControl.Destination = DestinationTypeEnum.Eject;
            job.SmartCardDataOnly(1, out actionID);
            while (true)
            {
                alarm = job.GetJobStatus(actionID, out uuidJob,
                    out printingStatus, out errorCode,
                    out copiesCompleted, out copiesRequested,
                    out magStatus, out contactStatus,
                    out contactlessStatus);
                if (contactStatus == "at_station")
                    break;
            }

            // PC/SC smart card code goes here
            // pseudocode example
            if smart card encoding is successful
                job.JobResume();
            else
                jobs can be aborted, job.JobAbort(),
                or retried, job.JobRetry()

            while (true)
            {
                alarm = job.GetJobStatus(actionID, out uuidJob,
                    out printingStatus, out errorCode,
                    out copiesCompleted, out copiesRequested,
                    out magStatus, out contactStatus,
                    out contactlessStatus);
                if (printingStatus == "done_ok" ||
                    printingStatus == "cleaning_up" ||
                    printingStatus == "done_error" ||
                    printingStatus == "cancelled_by_user" ||
                    printingStatus == "cancelled_by_error") break;
            }
            if error status
                jobs can be aborted or retried
        }
        catch (Exception ex)
        {
            errMsg = ex.Message;
        }
    }

```

## GetJobList

Description: Gets a list of the pending jobs.

Syntax:        short GetJobList( out object jobList )

Parameters:   jobList                    [out]string array contains "actionID,  
  uuid, status"

Example:        "ActionID: 13, UUID: 86b8d6bc-66f8-4758-acd9-7a3036901094,  
                      Status: done"  
                  "ActionID: 14, UUID: 593839fc-f889-44d0-8df4-1ee0880695f9,  
                      Status: in\_progress"

Returns:        error code (see [Appendix A](#))

Sample:        try  
                  {    object objJobList = null;  
                      alarm = job.GetJobList(out objJobList);  
                      if (objJobList != null)  
                      {    Array array        = (Array)objPrinterList;  
                              string[] jobList = new string[array.GetLength(0)];  
                              for (int i = 0; i < array.GetLength(0); i++)  
                                      jobList[i] = (string)array.GetValue(i);  
                      }  
                  }  
                  catch (Exception ex)  
                  {    errMsg = ex.Message;  
                      }  
                  }

## GetJobStatus

Description: Gets job status.

Syntax: short GetJobStatus(

```

    int          actionID,
    out string    uuidJob,
    out bool      readyForNextJob,
    out string    printingStatus,
    out string    cardPosition,
    out int       errorCode,
    out int       copiesCompleted,
    out int       copiesRequested,
    out string    magStatus,
    out string    contactStatus,
    out string    contactlessStatus)

```

Parameters:

actionID	[in ]job's Action ID
uuidJob	[out]job's UUID
readyForNextJob	[out] indicates whether or not the printer can accept a new job true = can accept new job false = cannot accept new job
printingStatus	[out]present job status
cardPosition	[out]card's current position in printer
errorCode	[out]error code
copiesCompleted	[out]number of copies complete
copiesRequested	[out]number of copies requested
magStatus	[out]magnetic encoding status
contactStatus	[out]contact encoding status
contactlessStatus	[out]contactless encoding status

- printingStatus:
 

"initializing",	"receiving",	"receive_ok",
"receive_error",	"receive_offline",	"parsed",
"in_progress",	"done_ok",	"done_error",
"cancelled_by_user",	"cancelled_by_error",	"cleaning_up",
- cardPosition:
 

"not_in_printer",	"held",	"ejecting_eject",
"ejecting_reject",	"ejecting_feeder"	"lamineate_insert",
- magStatus:
 

"encoding",	"verifying",	"reading",
"read_error",	"read_ein_error",	"write_error",
"retrace_error"		
- contactStatus:
 

"at_station",	"encoding",	"smart_encode_error", "contact_error"
---------------	-------------	---------------------------------------
- contactlessStatus:
 

"at_station",	"encoding",	"smart_encode_error",
"contactless_error"		

Returns: error code (see [Appendix A](#))

Sample: See next page

```
Sample 1:  while (true)
            {   alarm = job.GetJobStatus(actionID, out uuidJob,
                out printingStatus, out errorCode, out copiesCompleted,
                out copiesRequested, out magStatus, out contactStatus,
                out contactlessStatus);
            if ( ... check status condition ...)
                break;
            }

Sample 2:  Short alarm = job.GetJobStatus(actionID, out uuidJob,
            out printingStatus, out cardPosition, out errorCode,
            out copiesCompleted, out copiesRequested, out magStatus,
            out contactStatus, out contactlessStatus );
```

## GetJobStatusEx

Description: Gets job status.

Syntax: short GetJobStatus(

```

    int                actionID,
    out string          uuidJob,
    out bool            readyForNextJob,
    out string          printingStatus,
    out string          cardPosition,
    out int             errorCode,
    out int             copiesCompleted,
    out int             copiesRequested,
    out string          Status,
    out string          contactStatus,
    out string          contactlessStatus)

```

Parameters:

actionID	[in ]job's Action ID
uuidJob	[out]job's UUID
readyForNextJob	[out] indicates whether or not the printer can accept a new job
	true = can accept new job false = cannot accept new job
printingStatus	[out]present job status
cardPosition	[out]card's current position in printer
errorCode	[out]error code
copiesCompleted	[out]number of copies complete
copiesRequested	[out]number of copies requested
magStatus	[out]magnetic encoding status
contactStatus	[out]contact encoding status
contactlessStatus	[out]contactless encoding status

- printingStatus:
 

"initializing",	"receiving",	"receive_ok",
"receive_error",	"receive_offline",	"parsed",
"in_progress",	"done_ok",	"done_error",
"cancelled_by_user",	"cancelled_by_error",	"cleaning_up",
- cardPosition:
 

"not_in_printer",	"held",	"ejecting_eject",
"ejecting_reject",	"ejecting_feeder",	"laminate_insert"
- Status:
 

"encoding",	"verifying",	"reading",
"read_error",	"read_ein_error",	"write_error",
"retrace_error"		
- contactStatus:
 

"at_station",	"encoding",
"contact_error"	"smart_encode_error",
- contactlessStatus:
 

"at_station",	"encoding",
"contactless_error"	"smart_encode_error",

Returns: error code (see [Appendix A](#))

Sample: See next page

```
Sample 1:  while (true)
            {
                alarm = job.GetJobStatus(actionID, out uuidJob,
                                           out readyForNextJob,
                                           out printingStatus,
                                           out errorCode,
                                           out copiesCompleted,
                                           out copiesRequested,
                                           out magStatus,
                                           out contactStatus,
                                           out contactlessStatus);

                if ( ... check status condition ...)
                    break;
            }
```

## JobAbort

Description: Aborts a suspended job.

Syntax: `void JobAbort( boolean reject, out short alarm)`

Parameters: `reject` [in ]rejects a card if true  
`alarm` [out]error code (see [Appendix A](#))

Returns: `nothing`

Sample: `job.JobAbort(true, out alarm);`

## JobCancel

Description: Cancels a pending job.

Syntax:        `void JobCancel( int actionID )`

Parameters:   `actionID`                      [in ]a job's identifier

Returns:        `nothing`

Note:           `actionID = 0` cancels all jobs

Sample:        `job.JobCancel ( actionID ) ;`

## JobReprint

Description: Reprints the last job.

Syntax: `void JobReprint( int copies )`

Parameters: `copies` [in ]number of copies to reprint

Returns: `nothing`

Sample: `job.JobReprint ( copies );`

## JobResume

Description: Resumes a suspended job.

Syntax: `void JobResume()`

Parameters: nothing

Returns: nothing

Sample: `job.JobResume();`

## JobRetry

Description: Retries a suspended job.

Syntax: `void JobRetry()`

Parameters: nothing

Returns: nothing

Sample: `job.JobRetry();`

## SmartCardRetry

Description: Retries the last smart operation.

Syntax:       void SmartCardRetry(  
  SmartCardTypeEnum       scType )

Parameters:   scType               [in ]Smart Card Type (see [Appendix B](#) for  
  enumeration values)

Returns:       nothing

Sample:       job.SmartCardRetry(scType);

## ClearError

Description: Clears an error on a ZMotif device.

Syntax:        short ClearError()

Parameters:    nothing

Returns:       error code (see [Appendix A](#))

Sample:        alarm = job.ClearError();

## Reset

Description: Orders a printer to perform a reset.

```
Syntax:      short Reset (
                                ResetTypeEnum resetType )
```

Parameters:    resetType                    [in ]full or warm (see [Appendix B](#) for enumeration values)

Returns: error code (see [Appendix A](#))

Sample:      `alarm = job.Reset(ResetTypeEnum.Full);`

# GetPrinters

Description: Gets a list of available printers connected via USB or Ethernet.

```
Syntax:      void GetPrinters(
                ConnectionTypeEnum    conType,
                out object              printerList )
```

```
Parameters:  conType           [in ]connection type to search (see Appendix B for
           enumeration values)
           printerList        [out]printer list, string array
```

Returns: nothing

Note: In case of an Ethernet connected printer, the printer name will be followed by a comma and its corresponding IP Address ( "Zebra Printer Name, 10.1.4.82" )

```
Sample:      job = new Job();
             try
             {
                 object objPrinterList = null;
                 job.GetPrinters(ConnectionTypeEnum.USB, out objPrinterList);
                 if (objPrinterList != null)
                 {
                     Array array      = (Array)objPrinterList;
                     string[] prnList = new string[array.GetLength(0)];
                     for (int i = 0; i < array.GetLength(0); i++)
                         prnList[i] = (string)array.GetValue(i);
                 }
             }
             catch (Exception ex)
             {
                 string errMsg = ex.Message;
             }
             job = null;
```

## GetBroadcastConfiguration

Description: Gets Ethernet broadcasting configuration.

Syntax:           void GetBroadcastConfiguration(  
  out int                                 retries,  
  out float                            timeout,  
  out int                                 maxDevices )

Parameters:   retries                                 [out]number of times to broadcast  
              timeout                                 [out]timeout in seconds  
              maxDevices                             [out]max number of devices allowed

Returns:       nothing

Sample:        job.GetBroadcastConfiguration(out retries, out timeout,  
  out maxDevices);

## SetBroadcastConfiguration

Description: Sets Ethernet broadcasting configuration.

Syntax:       void SetBroadcastConfiguration(  
  int                              retries,  
  float                          timeout,  
  int                             maxDevices )

Parameters:   retries                   [in ]number of times to broadcast  
              timeout                  [in ]timeout in seconds  
              maxDevices               [in ]max number of devices allowed

Returns:       nothing

Sample:       job.SetBroadcastConfiguration(retries, timeout, maxDevices);

## ReserveDevice

Description: Reserves a device, used in shared environments.

Syntax:        short ReserveDevice(  
  ReservationTypeEnum        reservationType,  
  out int                      reservationToken)

Parameters:   reservationType    [in ]immediate or pending (see [Appendix B](#) for  
   enumeration values)  
                 reservationToken [out]reservation status

Returns:        error code (see [Appendix A](#))

Sample:        alarm = job.ReserveDevice(ReservationTypeEnum.ImmediateSession,  
  out reservationToken);

## GetReservationStatus

Description: Resturns the status of a reservation request.

Syntax:        void GetReservationStatus(  
   int                                reservationToken,  
   out bool                           sessionGranted )

Parameters:   reservationToken   [in ]reservation token for request status  
                 sessionGranted    [out]flag indicating reservation status:  
                                     true = reservation granted for token  
                                     false = reservation not granted for token

Returns:        error code (see [Appendix A](#))

## IsDeviceInSession

Description: Indicates if a device is in session.

Syntax:        short IsDeviceInSession(  
   int                                reservationToken,  
   out boolean                        inSession )

Parameters:   reservationToken   [in ]reservation status  
                 inSessionif        [out]true, device is in session

Returns:        error code (see [Appendix A](#))

Sample:        alarm = job.IsDeviceInSession(reservationToken, out inSession);

## ReleaseDevice

Description: Releases a reserved device.

Syntax:        short ReleaseDevice( int reservationToken )

Parameters:   reservationToken   [in ]reservation status

Returns:       error code (see [Appendix A](#))

Sample:       alarm = job.ReleaseDevice(reservationToken);

## GetSDKVersion

Description: Gets the SDK version.

Syntax:       void GetSDKVersion(  
                                    out byte                    major,  
                                    out byte                    minor,  
                                    out byte                    build,  
                                    out byte                    revision )

Parameters:   major               [out]major number of SDK version  
              minor               [out]minor number of SDK version  
              build               [out]build number of SDK version  
              revision            [out]revision number of SDK version

Returns:       nothing

Sample:       job.GetSDKVersion( out major, out minor, out build,  
                                  out revision );

## GetSDKProductVersion

Description: Gets the SDK product version, adheres to Zebra versioning standards.

Syntax: `void GetSDKProductVersion( out string productVersion )`

Parameters: `productVersion` [out]product version string

Returns: nothing

Sample: `string productVersion = string.Empty; job.GetSDKProductVersion( out productVersion );`

## EjectCard

Description: Sends card from internal hold position to eject bin.

Syntax: `EjectCard()`

Parameters: nothing

Returns: error code (see [Appendix A](#))

Sample: `Short Alarm = Job.EjectCard();`

## RejectCard

Description: Sends card from internal hold position to reject bin.

Syntax: `RejectCard()`

Parameters: nothing

Returns: error code (see [Appendix A](#))

Sample: `Short Alarm = Job.RejectCard();`

## ReturnCardToFeeder

Description: Sends card from internal hold position to feeder location.

Syntax:       ReturnCardToFeeder()

Parameters:   nothing

Returns:       error code (see [Appendix A](#))

Sample:       Short Alarm = Job.ReturnCardToFeeder();

## PositionCard

Description: Moves card from current location to the specified destination.

Syntax: `PositionCard( out int actionID )`

Parameters: `actionID` [out]returned by a ZMotif device identifying a job's ID

Returns: error code (see [Appendix A](#))

Sample 1: `Short Alarm = Job.PositionCard(out actionID);`

Note: The card's current location is defined by the `JobControl.FeederSource` attribute. The specified destination is defined by the `JobControl.Destination` attribute. These two parameters must be set prior to calling `PositionCard`.

The following sample code demonstrates how to move a card from the Card Feeder Hopper to the Eject Bin:

Sample 2:

```
//define the card's current location and specified destination:
Job.JobControl.FeederSource = FeederSourceEnum.CardFeeder
Job.JobControl.Destination = DestinationTypeEnum.Eject;

//move the card from the card feeder to the eject bin:
int actionID = 0;
short alarm = Job.PositionCard(out actionID);
```

## ReadBarcodeData

Description: Reads a pre-printed barcode from a card and returns the data.

Syntax:        short SendCommand(  
    out string                        barcodeType,  
    out string                        barcodeData,  
    out int                             actionID)

Parameters:   barcodeType            [out] type of barcode printed on the card barcodeData[out]  
                  data read from the barcode  
                  actionID                [out] job identifier associated with the  
                  barcode read operation

Return:        Error code - See [Appendix A](#)

Sample:        string barcodeType = string.Empty;  
                  string barcodeData = string.Empty;  
                  int actionID = 0;  
  
                  short alarm = job.ReadBarcodeData(out barcodeType,  
                  out barcodeData, out actionID);



# Job Control

## Properties

<u>Boolean</u>	
DeleteAfter	gets / sets if a job is to be deleted on completion
EnableOCPReprint	gets/sets a flag to enable/disable the printer's OCP panel's reprint button
MagVerification	gets / sets verification after magnetic encoding
<u>Enumeration</u>	
DataSource	gets / sets DataSourceEnum
Destination	gets / sets DestinationTypeEnum
FeederSource	gets / sets FeederSourceEnum
OrientationBack	gets / sets OrientationEnum
OrientationFront	gets / sets OrientationEnum
PrintOptimizationMode	gets / sets the PrintOptimizationModeEnum for ZXP Series 9 Printers
RotationBack	gets / sets RotationEnum
RotationFront	gets / sets RotationEnum
SharpnessLevelBack	gets / sets SharpnessLevelEnum
SharpnessLevelFront	gets / sets SharpnessLevelEnum
<u>Integer</u>	
CardThickness	gets / sets card thickness in mils default is 30
<u>Interface</u>	
CustomMagSettings	returns the CustomMagSettings Interface (see <a href="#">Section 8</a> )

Short	
ColorPreheatLevel	gets /sets the preheat level value for color printing
KPreheatLevelBack	gets /sets the preheat level value for k panel printing on the back-side of a card
KPreheatLevelFront	gets /sets the preheat level value for k panel printing on the front-side of a card
String CardType	gets / sets card type name; default is "PVC" (refer to <b>&lt;X-refBlue&gt;Section</b> for a list of available card types)PassPhrase sets the pass phrase in use by the printer refer to Toolbox Manual for details regarding the assignment of passphrase host authentication key and data encryption key to the printer)

## Methods

JobConfiguration .....	102
SideConfiguration .....	103
MagConfiguration .....	104
SmartCardConfiguration .....	105
ConfigureDataEncryption .....	106
GetDataEncryptionConfiguration .....	107
GetAvailableCardTypes .....	108
GetCardTypeInfoInformation .....	109
IntegerFromColorName .....	110
SetKPanelOptimization .....	111
KPanelOverwrite .....	112
SetMonoConvType .....	113

## JobConfiguration

Description: Sets job properties.

Syntax:       void JobConfiguration (

	FeederSourceEnum	feeder
	string	cardType
	int	cardThickness
	DataSourceEnum	dataSrc
	boolean	deleteJob )

Parameters:   feeder               [in ]sets property FeederSource  
               cardTypesets       [in ]property CardType  
               cardThickness       [in ]sets property CardThickness  
               dataSrc            [in ]sets property DataSource  
               deleteJob          [in ]sets property DeleteAfter

Returns:       nothing

Sample:       job.JobControl.JobConfiguration(feeder cardType cardThickness  
   dataSrc deleteJob);

## SideConfiguration

Description: Sets side configuration properties.

Syntax:        void SideConfiguration (

	SideEnum	side
	OrientationEnum	imgOrientation
	RotationEnum	imgRotation
	SharpnessLevelEnum	imgSharpness )

Parameters:    side                    [in ]front back

imgOrientation	[in ]landscape portrait
imgRotation	[in ]0 180
imgSharpness	[in ]sets the image sharpness in the device

Returns:        nothing

Sample:        job.JobControl.SideConfiguration(side imgOrientation  
imgRotation imgSharpness);

## MagConfiguration

Description: Sets magnetic encoding properties.

Syntax:       void MagConfiguration (

	SideEnum	side
	MagCoercivityEnum	coercivity
	MagEncodingTypeEnum	encodingType
	MagDataFormatEnum	format
	boolean	verify )

Parameters:   side               [in ]front back

coercivity	[in ]sets property MagCoercivity
encodingType	[in ]sets property MagEncodingType
format	[in ]sets property MagDataFormat
verify	[in ]indicates if verify after encoding

Returns:       nothing

Sample:       job.JobControl.MagConfiguration(side coercivity encodingType  
  format verify);

## SmartCardConfiguration

Description: Indicates if smart card encoding is part of a job.

```
Syntax:      void SmartCardConfiguration (
                                SideEnum          side
                                SmartCardTypeEnum smartCardType
                                boolean            program )
```

Parameters:	side	[in ]side the smart card chip is on
	smartCardType	[in ]type of smart card
	program	[in ]true indicates that the job will suspend for smart card encoding

Returns: nothing

```
Sample:      job.JobControl.SmartCardConfiguration(side smartCardType
              program);
```

## ConfigureDataEncryption

Description: Sets what will be encrypted.

[illegible]

Parameters:	encryptionType	[in ]sets the type of encryption
	encJobControl	[in ]identifies if the job control is to be encrypted
	encGraphics	[in ]identifies if the graphic image is to be encrypted

Returns: nothing

```
Sample:      job.JobControl.ConfigureDataEncryption(EncryptionTypeEnum.AES
              false false);
```

## GetDataEncryptionConfiguration

Description: Gets what will be encrypted.

Syntax:        void GetDataEncryptionConfiguration (

	out EncryptionTypeEnum	encryptionType
	out boolean	encJobControl
	out boolean	encGraphics )

Parameters:   encryptionType        [out]type of encryption

encJobControl	[out]identifies if the job control is to be encrypted
encGraphics	[out]identifies if the graphic image is to be encrypted

Returns:        nothing

Sample:        job.JobControl.GetDataEncryptionConfiguration (out  
                 encryptionType encJobControl encGraphics);

## GetAvailableCardTypes

Note • This Method applies to ZXP Series 8 Printers only.

Description: Gets a list of supported card types.

Syntax: `void GetAvailableCardTypes ( out object cardTypes )`

Parameters: `cardTypes` [out]array of card types returned as an object

Returns: nothing

```
Sample:
try
{
    job.JobControl.GetAvailableCardTypes(out objCardTypes);
    if (objCardTypes != null)
    {
        Array array = (Array)objCardTypes;
        string[] cardTypes = new string[array.GetLength(0)];
        for (int i = 0; i < array.GetLength(0); i++)
            cardTypes[i] = (string)array.GetValue(i);
    }
}
catch (Exception ex)
{
    errMsg = ex.Message;
}
```

## GetCardTypeInfoInformation

Description: Gets information on a card type.

[illegible]

Parameters:	cardTypes	[in ]card type
	xmlCardData	[out]XML file containing card information for the card type selected

Returns: nothing

```
Sample:      job.JobControl.GetCardTypeInfoInformation(cardType
              out xmlCardData);
```

## IntegerFromColorName

Description: Gets the integer value for a color.

Syntax:        `int IntegerFromColorName( string colorName )`

Parameters:   `colorName`                `[in ]color name`

Returns:       `integer value which represents the color name's color value`

## SetKPanelOptimization

Description: Sets the K panel optimization for the front or back panel.

Syntax:       void SetKPanelOptimization ( SideEnum side )

Parameters:    side                    [in ]K panel side for optimization (see [Appendix B](#) for enumeration values)

Returns: nothing

Sample: `job.JobControl.SetKPanelOptimization(SideEnum.Front)`

## KPanelOverwrite

Description: Enables k panel overwrite on a side of a card.

Syntax:       void KPanelOverwrite(  
   SideEnum                       side  
   bool                            Enabled );

Parameters:   side                       [in ]the side of the card to enable/disable  
    the k-panel overwrite (see [Appendix B](#) for  
    enumeration values)

                 Enabled               [in ]flag indicating whether or not k-panel  
    overwrite is enabled  
    true = k-panel overwrite enabled  
    false = k-panel overwrite disabled

Returns:       none

Sample:        //enable k-panel overwrite feature for the front-side of a card:  
                  SideEnum side = SideEnum.Front;  
                  bool Enabled = true;  
                  job.JobControl.KPanelOverwrite(side Enabled);

## SetMonoConvType

**Description:** Defines the type of monochrome conversion to be performed for the designated side of a card.

**Syntax:**

```
void SetMonoConvType(
    SideEnum          side
    MonoConvTypeEnum  convType);
```

**Parameters:**

side	[in ]the side of the card to enable/disable the k-panel overwrite (see <a href="#">Appendix B</a> for enumeration values)
convType	[in ]the type of monochrome conversion to be used (see <a href="#">Appendix B</a> for enumeration values)

**Returns:** error code (see [Appendix A](#))

**Sample:**

```
//enable monochrome conversion for the back-side of a card:
SideEnum side = SideEnum.Back;

//barcode to be converted
MonoConvTypeEnum1 convType = MonoConvTypeEnum.MonoBarcode;

job.JobControl.SetMonoConvType(side convType);
```



# Device

## Properties

### Boolean

AuthenticationEnabled	indicates if authentication is active read only
EncryptionEnabled	indicates if encryption is active read only
HasBarcode	returns true if the printer configuration includes a barcode readermodule.
HasLaminator	indicates if printer configuration includes a laminator read only
IsPrinterLocked	indicates if printer is locked (true = locked false = unlocked)
SNMPEnabled	enables/disables the SNMP protocol on the printer

### Byte

LCDContrast	sets the LCD display value
-------------	----------------------------

### Enumerations

ActiveLUTTable	returns the LUT table type that is currently in use by the printer
ErrorControlLevel	sets/gets the ErrorControlLevelEnum
EventLogLevel	gets / sets the level of logging used for the event log entries
ImageTransferType	gets the TransferTypeEnum
OCPLanguage	sets/gets the OCPLanguageEnum
PrintCapability	gets the TransferTypeEnum
StandbyTimeout	sets/gets the StandbyTimeoutEnum
USBSpeed	sets/gets the USBSpeedEnum

### Integer

CleanWarningThreshold	sets/gets the threshold value which will trigger the printer to display the clean printer message on the OCP.
HeadResistance	sets the head resistance default = 0
ParamChgCounter	gets the number of times parameters have been changed
SmartCardOffset	sets the smart card x offset position value is set during production

USBReadTimeout	assigns the timeout value for reading data from the USB port
USBWriteTimeout	assigns the timeout value for writing data to the USB port

### **Object**

PrinterEncryptionKey	gets /sets the encryption key to be used
PrinterLockKey	assigns the key for the printer's lock command (must be 32-byte hexadecimal value)

### **Short**

HttpPort	gets / sets the port number used for HTTP
----------	---

### **String**

PrinterType	gets the type of printer read only
WirelessMACAddress	gets /sets the host's MAC Address for wireless access

## Methods

GetPrinterStatus .....	118
ResumeFromStandby .....	119
GetCapabilities .....	120
GetConfiguration .....	121
SetConfiguration .....	122
GetCustomCardConfiguration .....	123
SetCustomCardConfiguration .....	124
GetAvailableMemory .....	125
GetDeviceInfo .....	126
GetFilmParams .....	127
GetLog .....	128
GetMagneticEncoderConfiguration .....	129
GetMagStartSentinelOffset .....	130
SetMagStartSentinelOffset .....	131
GetNetworkParams .....	132
SetNetworkParams .....	133
GetPanelPowerLevels .....	134
GetPanelPowerLevelsEx .....	135
SetPanelPowerLevels .....	136
SetPanelPowerLevelsEx .....	137
GetRejectCardCount .....	138
ClearRejectBinCount .....	139
GetRibbonParams .....	140
GetSensorStates .....	141
GetSensorValues .....	142
GetSmartCardConfiguration .....	143
GetStatusMessageString .....	144
GetTotalCardCount .....	145
GetTransferTempOffsets .....	146
SetTransferTempOffsets .....	147
GetDisplayedOCPPMessage .....	148
BuildOCPPMessage .....	149
ClearOCPPMessage .....	150
DisplayOCPPMessage .....	151
SelectOCPPButton .....	152
GetCleaningIntervals .....	153
SetCleaningIntervals .....	154
UpgradeFirmware .....	155
LockPrinter .....	156
UnlockPrinter .....	157
GetNVMEMData .....	158
SetNVMEMData .....	159
GetWirelessStatus .....	160
GetWirelessRadioStatus .....	161
GetWirelessParams .....	162
SetWirelessParams .....	164
ScanWirelessAccessPoints .....	165
GetCalibrationTableData .....	166
GetCalibrationTableNames .....	167
GetClock .....	168
GetI2CErrorStats .....	169
LoadCalibrationTableData .....	170
SetClock .....	171

## GetPrinterStatus

Description: Gets printer status.

Syntax:        short GetPrinterStatus (

out string	status
out int	error
out int	jobsPending
out int	jobsActive
out int	jobsComplete
out int	jobErrors
out int	jobsTotal
out int	nextActionID )

Parameters:    status            [out]status message  
                  error            [out]error code value  
                  jobsPending      [out]number of jobs in the printer's queue  
                  jobsActive       [out]number of active jobs  
                  jobsComplete     [out]number of jobs completed  
                  jobErrors        [out]number of job errors  
                  jobsTotaltotal   [out]number of jobs processed  
                  nextActionID     [out]next job's action ID

Status:        "initializing" "idle" "xfer\_rollers\_heating" "standby"  
                  "printing" "printing\_heating" "alarm\_handling" "offline"  
                  "canceling" "temp\_out\_of\_range" "mag\_ops" "contact\_ops"  
                  "contactless\_ops" "config\_data" "job\_data" "diagnostic\_mode"  
                  "xfer\_rollers\_cooling" "printing\_cooling" "insert\_card"

Returns:       Error code (see [Appendix A](#))

Sample:        alarm = job.Device.GetPrinterStatus(out status out error  
                                  out jobsPending out jobsActive out jobsCompleted  
                                  out jobErrors out jobsTotal out nextActionID);

## ResumeFromStandby

Description: Resumes operations after a printer is in standby mode.

Syntax:        `void ResumeFromStandBy ( void )`

Parameters:    `nothing`

Returns:       `nothing`

## GetCapabilities

Description: Gets the capabilities xml document.

```
Syntax:      short GetCapabilities (
                                CapabilitiesReportTypeEnum  reportType
                                out string                    xmlCapabilities )
```

Parameters:	reportType	[in ]general or media
	xmlCapabilities	[out]document containing device capability information

Returns: Error code (see [Appendix A](#))

Note: See [Appendix C](#) for an example of an XML Capabilities Document.

```
Sample:      alarm = job.Device.GetCapabilities(CapabilitiesReportTypeEnum.  
            General out xmlCapabilities);
```

## GetConfiguration

Description: Gets the xml configuration document.

Syntax:      short GetConfiguration ( out string xmlConfig )

Parameters:   xmlConfig                   [out]document containing configuration information

Returns: Error code (see [Appendix A](#))

Note: See [Appendix C](#) for an example of an XML Configuration Document.

Sample:      `alarm = job.Device.GetConfiguration(out xmlConfig);`

## SetConfiguration

Description: Updates the xml configuration document.

Syntax:        short SetConfiguration ( string xmlConfig )

Parameters:   xmlConfig           [in ]document containing configuration information

Returns: Error code (see [Appendix A](#))

Note: See [Appendix C](#) for an example of an XML Configuration Document.

Sample:      `alarm = job.Device.SetConfiguration( xmlConfig);`

## GetCustomCardConfiguration

Description: Gets the XML custom card configuration document.

[illegible]

```
Parameters:  cardNumber      [in ]custom card number
             xmlCardConfig   [out]document containing the custom card
                               configuration information
```

Returns: Error code (see [Appendix A](#))

Note 1: See [Appendix C](#) for an example of an XML Card Configuration Document.

Note 2: The custom card number can only be 1 or 2; this corresponds to Custom Card 1 and Custom Card 2 respectively.

```
Sample:      alarm = job.Device.GetCustomCardConfiguration(cardNumber
              out xmlCardConfig);
```

## SetCustomCardConfiguration

Description: Sets custom card configuration.

Syntax:        `short SetCustomCardConfiguration ( string xmlCardConfig )`

Parameters:   `xmlCardConfig`        [in ]document specifying custom card  
   configuration information

Returns:       Error code (see [Appendix A](#))

Note:           See [Appendix C](#) for an example of an XML Custom Card Configuration Document.

Sample:        `alarm = job.Device.SetCustomCardConfiguration(xmlCardConfig);`

## GetAvailableMemory

Description: Gets available device memory size.

```
Syntax:      short GetAvailableMemory (
                                MemorySourceEnum memType
                                out int memSize )
```

```
Parameters:  memType      [in ]RAM Flash
             emSize       [out]byte count of free memory
```

Returns: Error code (see [Appendix A](#))

```
Sample:    alarm = job.Device.GetAvailableMemory(MemorySourceEnum.MemRAM
           out memSize);
```

## GetDeviceInfo

Description: Gets device information.

```
Syntax:      short GetDeviceInfo (
                                out string      vendor
                                out string      model
                                out string      serialNumber
                                out string      MAC
                                out string      headSerialNumber
                                out string      oemCode
                                out string      fwVersion
                                out string      mediaVersion
                                out string      heaterVersion
                                out string      zmotifVersion )
```

```
Parameters:  vendor      [out]vendor name
              model      [out]model number
              serialNumber [out]serial number
              MAC         [out]MAC address
              headSerialNumber [out]print head serial number
              oemCode      [out]OEM model number
              fwVersion    [out]firmware version
              mediaVersion [out]media version
              heaterVersion [out]heater version
              zmotifVersion [out]ZMotif version number
```

Returns: Error code (see [Appendix A](#))

```
Sample:      alarm = job.Device.GetDeviceInfo(out vendor out model
        out serialNumber out MAC headSerialNumber
        out oemCode out fwVersion out mediaVersion
        out heaterVersion out zmotifVersion);
```

## GetFilmParams

Note • This Method applies to ZXP Series 8 Printers only.

Description: Gets film information.

```
Syntax:      short GetFilmParams (
                                out int           type
                                out string        partNumber
                                out string        description
                                out string        oemCode
                                out int           initialSize
                                out int           panelsRemaining )
```

```
Parameters:  type           [out]film type
              partNumber    [out]film part number
              description    [out]description of the film
              oemCode       [out]OEM part number
              initialSize    [out]total number of panels
              panelsRemaining [out]number of panels remaining
```

Retruns: Error code (see [Appendix A](#))

```
Sample:      alarm = job.Device.GetFilmParams(out type out partNumber
                                                out description out oemCode out initialSize
                                                out panelsRemaining);
```

## GetLog

Description: Gets XML log data.

```
Syntax:      short GetLog (
                                LogTypeEnum      logType
                                boolean            clear
                                out string        xmlLog )
```

Parameters:	logType	[in ]type of log
	clear	[in ]if true clears log after reading
	xmlLog	[out]document containing log information

Returns: Error code (see [Appendix A](#))

Note: Logs are maintained across power cycles.

```

Buffer requirements:
    LogTypeEnum.Errors: bufferSize ≥ 2048 bytes
    LogTypeEnum.Cleaning | LogTypeEnum.Service: bufferSize ≥ 256 bytes
    LogTypeEnum.Events: bufferSize ≥ 65535 bytes

```

```
Sample:      alarm = job.Device.GetLog(LogTypeEnum.Events clear
              out xmlLog);
```

## GetMagneticEncoderConfiguration

Description: Gets magnetic encoder configuration.

[illegible]

```
Parameters:  headType      [out]type of magnetic head
            stripeLocation [out]"top" or "bottom"
```

Returns: nothing

```
Sample:      job.Device.GetMagneticEncoderConfiguration(out headType
              out stripeLocation);
```

## GetMagStartSentinelOffset

Description: Returns the offset value of the start sentinel for a specific magnetic track.

[illegible]

Parameters:	trkNumber	[in ]the magnetic track (see <a href="#">Appendix B</a> for details)
	offset	[out]start sentinel offset for the magnetic track

Returns: Error code (see [Appendix A](#))

```
Sample:      MagTrackNumberEnum trkNumber = MagTrackNumberEnum.MagTrack1;
             short offset = 0;
             job.Device.GetMagStartSentinelOffset(trkNumber, out offset);
```

## SetMagStartSentinelOffset

Description: Sets the offset value of the start sentinel for a specific magnetic track.

[illegible]

Parameters:	trkNumber	[in ]the magnetic track (see <a href="#">Appendix B</a> for details)
	offset	[in ]start sentinel offset for the magnetic track

Returns: Error code (see [Appendix A](#))

```
Sample:    MagTrackNumberEnum trackNumber = MagTrackNumberEnum.MagTrack1;
           short offset = 10;
           job.Device.SetMagStartSentinelOffset(trackNumber, offset);
```

## GetNetworkParams

Description: Gets network parameters.

Syntax:        short GetNetworkParams (

out string	MAC
out string	ipAddress
out string	subMask
out string	gateway
out boolean	dhcpEnabled )

Parameters:    MAC                    [out]MAC address  
                 ipAddress            [out]TCP/IP address  
                 subMask              [out]submask address  
                 gateway              [out]gateway address  
                 dhcpEnabled          [out]indicates if dhcp is enabled or not

Returns:        Error code (see [Appendix A](#))

Sample:        alarm = job.Device.GetNetworkParams(out MAC out ipAddress  
   out subMask out gateway out dhcpEnabled);

## SetNetworkParams

Description: Sets network parameters.

```
Syntax:      short SetNetworkParams (
                                string      ipAddress
                                string      subMask
                                string      gateway
                                BoolTypeEnum dhcpEnabled )
```

Parameters:	ipAddress	[in ]network IP address for the device
	subMask	[in ]network submask address for the device
	gateway	[in ]gateway address for the device
	dhcpEnabled	[in ]BoolTypeEnum.NoChange BoolTypeEnum.False BoolTypeEnum.True

Returns: Error code (see [Appendix A](#))

Note: Any of the arguments can be null; if null property is not set.

```
Sample:      alarm = job.Device.SetNetworkParams(ipAddress subMask gateway
              BoolTypeEnum.NoChange);
```

## GetPanelPowerLevels

Description: Gets the panel power levels.

Syntax:        short GetPanelPowerLevels (

out byte	yellowPanel
out byte	magentaPanel
out byte	cyanPanel
out byte	kResinFront
out byte	kResinBack
out byte	kDye
out byte	overlayPanel
out byte	uvFront
out byte	uvBack )

Parameters:    yellowPanel        [out]yellow panel level  
                  magentaPanel     [out]magenta panel level  
                  cyanPanel        [out]cyan panel level  
                  kResinFront      [out]front monochrome panel level  
                  kResinBack      [out]back monochrome panel level  
                  kDye             [out]monochrome dye level  
                  overlayPanel     [out]overlay panel level  
                  uvFront         [out]front UV panel level  
                  uvBack          [out]back UV panel level

Returns:       Error code (see [Appendix A](#))

Sample:        alarm = job.Device.GetPanelPowerLevels(out yellowPanel  
                                  out magentaPanel out cyanPanel out kResinFront  
                                  out kResinBack out kDye out overlay out uvFront  
                                  out uvBack);

## GetPanelPowerLevelsEx

Description: Returns the panel power levels.

Syntax:        short GetPanelPowerLevelsEx (

out byte	yellowPanel
out byte	magentaPanel
out byte	cyanPanel
out byte	kResinFront
out byte	kResinBack
out byte	kDye
out byte	overlayPanel
out byte	uvFront
out byte	uvBack
out byte	inhibitPanel
out byte	helperPanel )

Parameters:    yellowPanel        [out]yellow panel level  
                   magentaPanel    [out]magenta panel level  
                   cyanPanel        [out]cyan panel level  
                   kResinFront      [out]front monochrome panel level  
                   kResinBack      [out]back monochrome panel level  
                   kDye             [out]monochrome dye level  
                   overlayPanel     [out]overlayPanel level  
                   uvFront          [out]front UV panel level  
                   uvBack           [out]back UV panel level  
                   inhibtPanel      [out]inhibitPanel level  
                   helperPanel      [out]helperPanel level

Returns:       Error code (see [Appendix A](#))

Sample:        alarm = job.Device.GetPanelPowerLevelsEx(out yellowPanel  
                   out magentaPanel out cyanPanel out kResinFront  
                   out kResinBack out kDye out overlayPanel out uvFront  
                   out uvBack out inhibitPanel out helperPanel);

## SetPanelPowerLevels

Description: Sets the panel power levels.

Syntax:           short SetPanelPowerLevels(

byte	yellowPanel
byte	magentaPanel
byte	cyanPanel
byte	kResinFront
byte	kResinBack
byte	kDye
byte	overlayPanel
byte	uvFront
byte	uvBack )

Parameters:   yellowPanel       [in ]yellow panel level  
               magenta Panel   [in ]magenta panel level  
               cyanPanel       [in ]cyan panel level  
               kResinFront      [in ]front monochrome panel level  
               kResinBack      [in ]back monochrome panel level  
               kDye           [in ]monochrome dye level  
               overlayPanel   [in ]overlay panel level  
               uvFront       [in ]front UV panel level  
               uvBack       [in ]back UV panel level

Returns:       Error code (see [Appendix A](#))

Note:          Power levels are between 0 to 255.

Sample:       alarm = job.Device.SetPanelPowerLevels(yellowPanel  
   magentaPanel cyanPanel kResinFront kResinBack kDye  
   overlay uvFront uvBack);

## SetPanelPowerLevelsEx

Description: Sets the panel power levels.

```
Syntax:      short SetPanelPowerLevelsEx (
                                byte          yellowPanel
                                byte          magentaPanel
                                byte          cyanPanel
                                byte          kResinFront
                                byte          kResinBack
                                byte          kDye
                                byte          overlayPanel
                                byte          uvFront
                                byte          uvBack
                                byte          inhibitPanel
                                byte          helperPanel )
```

Parameters:	yellowPanel	[in ]yellow panel level
	magentaPanel	[in ]magenta panel level
	cyanPanel	[in ]cyan panel level
	kResinFront	[in ]front monochrome panel level
	kResinBack	[in ]back monochrome panel level
	kDye	[in ]monochrome dye level
	overlayPanel	[in ]overlay panel level
	uvFront	[in ]front UV panel level
	uvBack	[in ]back UV panel level
	inhibitPanel	[in ]inhibitPanel level
	helperPanel	[in ]helperPanel level

Returns: Error code (see [Appendix A](#))

```
Sample:      alarm = job.Device.SetPanelPowerLevelsEx(yellowPanel
              agentaPanel cyanPanel kResinFront kResinBack kDye
              overlayPanel uvFront uvBack inhibitPanel helperPanel);
```

## GetRejectCardCount

**Description:** Returns the number of cards currently in the reject bin and the total number of cards that have been sent to the printer reject bin.

**Syntax:**

```
short GetRejectCardCount(  
    out int cardsInBin  
    out int cardsRejected)
```

**Parameters:**

cardsInBin	[out]the number of cards currently in the reject bin
cardsRejected	[out]the total number of cards sent to the reject bin

**Returns:** Error code (see [Appendix A](#))

**Sample:**

```
int cardsInBin = 0; int cardsRejected = 0;  
short alarm = job.Device.GetRejectCardCount(out cardsInBin  
out cardsRejected );
```

## ClearRejectBinCount

**Description:** Sets the counter for the number of cards currently in the reject bin to zero

**Syntax:** `short ClearRejectBinCount()`

**Parameters:** none

**Returns:** Error code (see [Appendix A](#))

**Sample:** `short alarm = job.Device.ClearRejectBinCount();`

## GetRibbonParams

Description: Gets ribbon information.

Syntax:        short GetRibbonParams (

out int	type
out string	partNumber
out string	description
out string	oemCode
out int	initialSize
out int	panelsRemaining )

Parameters:    type                    [out]type of ribbon  
                 partNumber            [out]ribbon part number  
                 description           [out]description of the ribbon  
                 oemCode                [out]OEM part number  
                 initialSize            [out]total number of panels  
                 panelsRemaining        [out]unused number of panels

Returns:       Error code (see [Appendix A](#))

Sample:        alarm = job.Device.GetRibbonParams(out type out partNumber  
    out description out oemCode out initialSize  
    out panelsRemaining);

## GetSensorStates

Note • Any reference to “film” applies to ZXP Series 8 Printers only.

Description: Gets device sensor states.

Syntax: `short GetSensorStates( out object sensorStates )`

Parameters: `sensorStates` [out]string array which contains the sensor states returned as an object

Example:

"FilmTakeupEncoder: unknown"	"RibbonTakeupEncoder: unknown"
"RibbonPayoutEncoder: unknown"	"DoorOpen: no"
"CardEdgeBlocked: no"	"TricolorState: k_front_panel"
"HeadCamBlocked: yes"	"FilmStripe1Blocked: no"
"FilmStripe2Blocked: no"	"CardFeederBlocked: yes"
"TricolorError: 0"	

Returns: Error code (see [Appendix A](#))

Sample:

```
try
{
    object objSensorStates = null;
    alarm = job.Device.GetSensorStates(out objSensorStates);
    if (objSensorStates != null)
    {
        Array array = (Array)objSensorStates;
        string[] sensorStates = new string[array.GetLength(0)];
        for (int i = 0; i < array.GetLength(0); i++)
        {
            sensorStates[i] = (string)array.GetValue(i);
            if (i == 0)
                sensorState = sensorStates[i];
        }
    }
}
catch (Exception ex)
{
    errMsg = ex.Message;
}
```

## GetSensorValues

Description: Gets device sensor values.

Syntax: `short GetSensorValues ( out object sensorValues )`

Parameters: `sensorValues` [out]string array which contains the sensor values returned as an object

Example:

"Voltage24: 24.150000"	"VoltageAC: 110"
"VoltageRaw: 744"	"MagTrack1: 516"
"MagTrack2: 516"	"MagTrack3: 515"
"PrintheadTemperature: 38"	"MagHeadType: 1001"
"RibbonBEMF: 515"	"TricolorAny: 971"
"TricolorRed: 864"	"TricolorGreen: 873"
"TricolorBlue: 977"	"TopTransferTemperature: 185"
"BottomTransferTemperature: 76"	

Returns: Error code (see [Appendix A](#))

Sample:

```
try
{
    object objSensorValues = null;
    alarm = job.Device.GetSensorValues(out objSensorValues);
    if (objSensorValues != null)
    {
        Array array = (Array)objSensorValues;
        string[] sensorValues = new string[array.GetLength(0)];
        for (int i = 0; i < array.GetLength(0); i++)
        {
            sensorValues[i] = (string)array.GetValue(i);
            if (i == 0)
                sensorValue = sensorValues[i];
        }
    }
}
catch (Exception ex)
{
    errMsg = ex.Message;
}
```

## GetSmartCardConfiguration

Description: Gets smart card configuration.

Syntax:           void GetSmartCardConfiguration(  
  out string                       commChannel  
  out string                       scContact  
  out string                       scContactless )

Parameters:   commChannel           [out]communication channel to communicate  
  with the smart card encoder  
                  scContact           [out]type of contact encoder  
                  scContactless       [out]type of contactless encoder

Returns:       nothing

Sample:        job.Device.GetSmartCardConfiguration(out commChannel  
  out scContact out scContactless);

## GetStatusMessageString

Description: Returns a status message string for a specific alarm or error code.

Syntax: `string GetStatusMessageString ( short statusCode )`

Parameters: `statusCode` [in ]alarm or error code

Returns: description of alarm or error code (see [Appendix A](#))

Sample: `message = job.Device.GetStatusMessageString(statusCode);`

## GetTotalCardCount

Description: Gets the total card count since last reset (secure action reset).

Syntax:        `short GetTotalCardCount ( out int cardCount )`

Parameters:   `cardCounttotal`        [out]number of cards printed since last reset

Returns:       Error code (see [Appendix A](#))

Sample:        `alarm = job.Device.GetTotalCardCount(out cardCount);`

## GetTransferTempOffsets

Description: Gets the heater temperature settings.

[illegible]

Parameters:	topSnglOffset	[out]top roller single side printing offset from manufacturing temperature setting
	topDb1Offset	[out]top roller double side printing offset from manufacturing temperature setting
	botDb1Offset	[out]bottom roller double side printing offset from manufacturing temperature setting

Returns: Error code (see [Appendix A](#))

```
Sample:      alarm = job.Device.GetTransferTempOffsets(out topSnglOffset
              out topDblOffset out botDblOffset);
```

## SetTransferTempOffsets

Description: Sets the heater temperatures.

```
Syntax:      short SetTransferTempOffsets (
                                int           topSnglOffset
                                int           topDblOffset
                                int           botDblOffset )
```

Parameters:	topSnglOffset	[in ]top roller single side printing offset from manufacturing temperature setting
	topDblOffset	[in ]top roller double side printing offset from manufacturing temperature setting
	botDblOffset	[in ]bottom roller double side printing offset from manufacturing temperature setting

Returns: Error code (see [Appendix A](#))

```
Sample:      alarm = job.Device.SetTransferTempOffsets(topSnglOffset
               botSnglOffset topDblOffset botDblOffset);
```

## GetDisplayedOCPMessage

Description: Retrieves the current OCP display.

Syntax: `void GetDisplayedOCPMessage ( out string ocpMessage )`

Parameters: `ocpMessage` [out ]text currently being displayed returned as xml document

Returns: nothing

Sample: `job.Device.GetDisplayedOCPMessage(out ocpMessage);`

Example:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<displayed_message>
  <text line="0" reverse="no" underline="no" double="yes" toggle="no" >ALARM </text>
  <text line="3" reverse="no" underline="no" double="no" toggle="no" >OUT OF CARDS </text>
  <text line="4" reverse="no" underline="no" double="no" toggle="no" >ERROR CODE = 4001 </text>
  <text line="6" reverse="no" underline="no" double="no" toggle="no" > MENU INFO RETRY </text>
</displayed_message>
```

## BuildOCPMessage

Description: Builds a line specific OCP message.

Syntax:        void BuildOCPMessage (

byte	lineNumber
string	message
boolean	underline
boolean	reverse
boolean	blinkmode
boolean	doubleSize )

Parameters:    lineNumber        [in ]OCP line number ( 1 - 7 )

                message         [in ]text to be displayed

                underline       [in ]if text is to be underlined

                reverse         [in ]if text is to be displayed in

                                 reverse mode

                blinkmode       [in ]                                false = display continuous text

   true = display blinking text

                doubleSize       [in ]if text is be be displayed double size

Returns:        nothing

Sample:        job.Device.BuildOCPMessage(lineNumber message underline

                 reverse toggle doubleSize);

## ClearOCPMessage

Description: Clears the OCP message panel.

Syntax:        short ClearOCPMessage ( boolean ocpLock )

```
Parameters:  ocpLock      [in ]true: freezes the display and disables
                        the buttons
                        [in ]false: allows display updates and
                        enables the buttons
```

Returns: Error code (see [Appendix A](#))

Sample:      `alarm = job.Device.ClearOCPMessage(ocpLock);`

## DisplayOCPPMessage

**Description:** Displays the message on the printer's OCP display (built by function BuildOCPPMessage).

**Syntax:** short DisplayOCPPMessage ( boolean ocpLock )

**Parameters:** ocpLock [in ]true: freezes the display and disables the buttons  
[in ]false: allows display updates and enables the buttons

**Returns:** Error code (see [Appendix A](#))

**Note:** After locking the OCP it is necessary to unlock it with ClearOCPPMessage()

**Sample:** alarm = job.Device.DisplayOCPPMessage(ocpLock);

## SelectOCPButton

Description: Allows a button on the printer's OCP display to be pressed via the SDK.

Syntax:        short SelectOCPButton ( OCPButtonEnum button )

Parameters:   button                      [in ]OCP button to be pressed.

Possible choices:

OCPButtonEnum.LeftButton  
OCPButtonEnum.MiddleButton  
OCPButtonEnum.RightButton

Returns:        Error code (see [Appendix A](#))

Sample:        alarm = job.Device.SelectOCPButton (OCPButton.LeftButton);

## GetCleaningIntervals

Description: Gets the cleaning intervals for the paths.

Syntax:           void GetCleaningIntervals (

out int	feederPath
out int	atmPath
out int	transferPath )

Parameters: feederPath           [out]cleaning intervals for feeder path  
            atmPath            [out]cleaning interval for ATM path  
            transferPath       [out]cleaning interval for transfer path

Returns:       nothing

Sample:        job.Device.GetCleaningIntervals(out feederPath out atmPath  
  out transferPath);

## SetCleaningIntervals

Description: Sets the paths cleaning intervals.

```
Syntax:      short SetCleaningIntervals (
                                int          feederPath
                                int          atmPath
                                int          transferPath )
```

Parameters:	feederPath	[in ]cleaning intervals for feeder path
	atmPath	[in ]cleaning interval for ATM path
	transferPath	[in ]cleaning interval for transfer path

Returns: Error code (see [Appendix A](#))

```
Sample:      alarm = job.Device.SetCleaningIntervals(feederPath atmPath
              transferPath);
```

## UpgradeFirmware

Description: Upgrades MCB HCB MAB firmware.

Syntax:        void UpgradeFirmware ( string firmwareFile )

Parameters:    firmwareFile            [in ]path and filename for the firmware

Returns:        nothing

Sample:        job.Device.UpgradeFirmware( firmwareFile )

Note:           Firmware is a binary file.

## LockPrinter

Description: Sends lock command to printer (financial configuration only).

Syntax:        short LockPrinter ( )

Parameters:   none

Returns:       Error code (see [Appendix A](#))

Sample:       short alarm = job.Device.LockPrinter();

## UnlockPrinter

Description: Sends unlock command to printer (financial configuration only).

Syntax:        `short UnlockPrinter ( int timeout )`

Parameters:   `timeout`                    [in ]number of seconds the printer will  
    remain unlocked after command is issued  
    with printer lid not physically opened.

Note:           Minimum number of seconds = 15  
                  Maximum number of seconds = 600

If a command is issued and the printer is not opened the printer will automatically relock itself after timeout has transpired.

Returns:        Error code (see [Appendix A](#))

Sample:        `short alarm = job.Device.UnlockPrinter(timeout);`

## GetNVMEMData

Description: Returns the printer's current non-volatile memory data as an xml document.

Note: This function is intended to be used as the data back-up operation of a back-up-and-restore operation only. The information returned is encrypted.

Syntax: `short GetNVMEMData( out string xmlNVMemData )`

Parameters: `xmlNVMemData` [out]xml xml document containing the non-volatile memory data

Returns: Error code (see [Appendix A](#))

Sample: 

```
string xmlNVMemData = string.Empty;
job.Device.GetNVMEMData(out xmlNVMemData);
```

## SetNVMEMData

Description: Restores the printer's non-volatile memory data.

Note: This function is intended to be used as the data restore operation of a back-up-and-restore operation only. The non-volatile memory data restored will be in the form of an xml document which was created using the GetNVMEMData function call.

Syntax: `short SetNVMEMData( string xmlNVMemData )`

Parameters: `xmlNVMemData` [in ]xml document containing the non-volatile memory data

Returns: Error code (see [Appendix A](#))

Sample: `string xmlNVMemData => xml document retrieved using GetNVMEMData  
short alarm = job.Device.SetNVMEMData(xmlNVMemData);`

## GetWirelessStatus

Description: Returns the status of the printer's wireless configuration.

Syntax: `short GetWirelessStatus( out object wirelessStatus );`

Parameters: `wirelessStatus` [out]20 element string array containing the wireless status

Returns: Error code (see [Appendix A](#))

```
Sample:
object wirelessStatus = null;
ArrayList lstWirelessStatus = null;
job.Device.GetWirelessStatus(out wirelessStatus);

//convert the object to an array:
lstWirelessStatus = ArrayList.Adapter((string[])wirelessStatus);

//example data:
[0] "State: connected"
[1] "SSID: CPS_DEMO"
[2] "Channel: 6"
[3] "ReceiveSignalStrength: -52"
[4] "SignalToNoiseRatio: 34"
[5] "NoiseFloor: -86"
[6] "Security: wpa2"
[7] "Crypto: ccmp"
[8] "Rate: 5"
[9] "BytesReceived: 65686"
[10] "UnicastPacketsReceived: 1123"
[11] "NonUnicastPacketsReceived: 0"
[12] "DiscardedPacketsReceived: 0"
[13] "ReceiveErrors: 0"
[14] "BytesSent: 11369"
[15] "UnicastPacketsSent: 60"
[16] "NonUnicastPacketsSent: 0"
[17] "DiscardedPacketsSent: 0"
[18] "SendErrors: 0"
[19] "LinkLost: 8"
```

## GetWirelessRadioStatus

Description: Returns the status of the printer's wireless radio configuration.

Syntax: `short GetWirelessRadioStatus( out object radioStatus );`

Parameters: `radioStatus` [out] 17 element string array containing the radio status

Returns: Error code (see [Appendix A](#))

```
Sample:
object radioStatus = null;
ArrayList lstRadioStatus = null;

job.Device.GetWirelessRadioStatus(out radioStatus);

//convert the object to an array:
lstRadioStatus = ArrayList.Adapter((string[])radioStatus);

//example data:
lstRadioStatus [0]"MulticastTXFrame: 38"
lstRadioStatus [1]"Failed: 0"
lstRadioStatus [2]"Retry: 5"
lstRadioStatus [3]"MultiRetry: 4"
lstRadioStatus [4]"FrameDuplicate: 3"
lstRadioStatus [5]"RTSSuccess: 0"
lstRadioStatus [6]"RTSFailure: 0"
lstRadioStatus [7]"ACKFailure: 18"
lstRadioStatus [8]"RxFrag: 11578"
lstRadioStatus [9]"MulticastRxFrame: 11510"
lstRadioStatus [10]"FrameCheckSeqError: 3349"
lstRadioStatus [11]"TxFrame: 54"
lstRadioStatus [12]"TxSuccessRate: 100"
lstRadioStatus [13]"TxRetryRate: 9"
lstRadioStatus [14]"RxSuccessRate: 77"
lstRadioStatus [15]"RxDuplicateRate: 0"
lstRadioStatus [16]"RTSSuccessRate: 0"
```

## GetWirelessParams

Description: Returns the current state of printer's wireless parameters.

Syntax:        short GetWirelessParams (

```

                                out bool          SNMPEnabled
                                out bool          dhcpEnabled
                                out string        ipAddress
                                out string        subMask
                                out string        gateway
                                out bool          radioEnabled
                                out WirelessSecurityTypeEnum wirelessSecurity
                                out WirelessCryptoEnum wirelessCrypto
                                out string        SSID
                                out string        bSSID
                                out byte         channel );
```

Parameters:    SNMPEnabled        [out]indicates whether or not SNMP protocol is enabled:  
                                       true - SNMP enabled  
                                       false - SNMP disabled

                DHCPEnabled        [out]indicates whether or not DHCP protocol is enabled:  
                                       true - DHCP enabled  
                                       false - DHCP disabled

                ipAddress         [out]the printer's ip address

                submask           [out]the subnet mask in use

                gateway           [out]the gateway in use

                radioEnabled[out]indicates whether or not the radio is enabled  
                                       true - radio enabled  
                                       false - radio disabled

                wirelessSecurity [out]the type of wireless security in use  
                                       (see [Appendix B](#) for details)

                wirelessCrypto [out]the type of encryption in use  
                                       (see [Appendix B](#) for details)

                SSID               [out]the service set identifier

                bSSID             [out]the basic service set identifier

                channel           [out]the type of channel in use

Returns:        Error code (see [Appendix A](#))

Sample:        See next page

```

Sample:    try
            {
                bool SNMPEnabled  = false;
                bool dhcpEnabled  = false;
                bool radioEnabled = false;

                byte channel = 0;

                string ipAddress  = string.Empty;
                string subMask    = string.Empty;
                string gateway    = string.Empty;
                string SSID       = string.Empty;
                string bSSID      = string.Empty;

                WirelessSecurityTypeEnum wirelessSecurity =
                    WirelessSecurityTypeEnum.Open;
                WirelessCryptoEnum wirelessCrypto = WirelessCryptoEnum.NoCrypto;
                short alarm = job.Device.GetWirelessParams(
                    out SNMPEnabled
                    out dhcpEnabled
                    out ipAddress
                    out subMask
                    out gateway
                    out radioEnabled
                    out wirelessSecurity
                    out wirelessCrypto
                    out SSID
                    out bSSID
                    out channel );

            }
            catch (Exception ex)
            {
                //handle exception
            }

```

## SetWirelessParams

Description: Sets the state of printer's wireless parameters.

```
Syntax: short SetWirelessParams(
    bool                SNMPEnabled
    bool                DHCPEnabled
    string              ipAddress
    string              subMask
    string              gateway
    bool                radioEnabled
    WirelessSecurityTypeEnum wirelessSecurity
    WirelessCryptoEnum  wirelessCrypto
    string              ssid
    string              bssid
    byte                channel);
```

Parameters:	SNMPEnabled	[in ]indicates whether or not SNMP protocol is enabled true - SNMP enabled false - SNMP disabled
	DHCPEnabled	[in ]indicates whether or not DHCP protocol is enabled true - DHCP enabled false - DHCP disabled
	ipAddress	[in ]the printer's ip address
	submask	[in ]the subnet mask in use
	gateway	[in ]the default gateway in use
	radioEnabled	[in]indicates whether or not the radio is enabled true - radio enabled false - radio disabled
	wirelessSecurity	[in ]the type of wireless security in use (see <a href="#">Appendix B</a> for details)
	wirelessCrypto	[in ]the type of encryption in use (see <a href="#">Appendix B</a> for details)
	ssid	[in ]the service set identifier
	bssid	[in ]the basic service set identifier
	channel	[in ]the wireless channel in use

Returns: Error code (see [Appendix A](#))

Sample: See next page.

```
Sample: BoolTypeEnum SNMPEnabled = BoolTypeEnum.True_BT;
        BoolTypeEnum DHCPEnabled = BoolTypeEnum.True_BT;
        string IPAddress = "100.100.100.100";
        string Submask = "255.255.255.0";
        string Gateway = "100.100.100.1";
        BoolTypeEnum radioEnabled = BoolTypeEnum.True_BT;
        WirelessSecurityTypeEnum wirelessSecurity = WirelessSecurityTypeEnum.Open;
        WirelessCryptoTypeEnum wirelessCrypto = WirelessCryptoTypeEnum.NoCrypto;
        string ssid = "DEMO_NETWORK";
        string bssid = "10:8c:cf:10:df:7a";
        object wirelesskey = keyVal;
        byte channel = 0;

        job.Device.SetWirelessParams(SNMPEnabled, DHCPEnabled,
                                     IPAddress, Submask, Gateway,
                                     radioEnabled, wirelessSecurity,
                                     wirelessCrypto, ssid, bssid,
                                     wirelessKey, channel);
```

## ScanWirelessAccessPoints

**Description:** Returns the access points discovered in an XML document.

```
Syntax:      short ScanWirelessAccessPoints(
                int                scanIterations
                out string          xmlAccessPoints);
```

```
Parameters:  scanIterations    [in ]number of iterations to use for scanning
             xmlAccessPoints   [out]XML document containing the access points
                               discovered
```

Returns: Error code (see [Appendix A](#))

[illegible][illegible]

```
//example xml document
<?xml version="1.0" encoding="UTF-8" ?>
<AccessPoints>
  <AccessPoint>
    <SSID>CPS_DEMO</SSID>
    <BSSID>10:8c:cf:10:d6:1a</BSSID>
    <Selected>no</Selected>
    <Channel>1</Channel>
    <Privacy>1</Privacy>
    <SignalStrength>62</SignalStrength>
    <Infrastructure>true</Infrastructure>
    <Security>WPA2</Security>
    <Encryption>CCMP</Encryption>
    <MaxDataRate>54</MaxDataRate>
  </AccessPoint>
  <AccessPoint>
    <SSID>ConfigRoom</SSID>
    <BSSID>00:24:d2:46:7b:27</BSSID>
    <Selected>no</Selected>
    <Channel>1</Channel>
    <Privacy>1</Privacy>
    <SignalStrength>47</SignalStrength>
    <Infrastructure>true</Infrastructure>
    <Security>WPA_WPA2</Security>
    <Encryption>TKIP_CCMP</Encryption>
    <MaxDataRate>54</MaxDataRate>
  </AccessPoint>

```

## GetCalibrationTableData

Description: Returns the selected calibration table's data.

Syntax:           short GetCalibrationTableData(  
   CalibrationTableEnum     table,  
   out string                 xmlCalibrationData)

Parameters:   table                                 [in ]type of calibration table being retrieved  
   See [Appendix B](#) for enumeration valuesxmlCalibrationData  
   [out]calibration data returned in xml format

Return:         Error code (see [Appendix A](#))

Sample:         string xmlCalibrationData = string.Empty;  
                   short alarm = job.Device.GetCalibrationTableData(  
   CalibrationTableEnum.LUT1Default,  
   out xmlCalibrationData);

## GetCalibrationTableNames

Description: Returns the default and installed calibration table names.

[illegible]

```
Parameters: defaultTableNames [out]string array of default table names
            returned as an object installedTableNames [out]string
            array of installed table names
            returned as an object
```

```
Sample:      object defaultTableNames = null;
            object installedTableNames = null;
            short alarm = job.Device.GetCalibrationTableNames(
                                out defaultTableNames, out installedTableNames);
            if(alarm == 0)//success
            {
                string[] defaultNames = (string[])defaultTableNames;
                string[] installedNames = (string[])installedTableNames;
            }
        }
```

## GetClock

Description: Returns the printer's current system clock value.

```
Syntax:      short GetClock(          out byte      month,
                                     out byte      day,
                                     out int       year,
                                     out byte      hour,
                                     out byte      minute)
```

```
Parameters:  month      [out] current month
             day        [out] current day
             year       [out] current year
             hour       [out] current hour
             minute     [out] current minute
```

Return: Error code (see [Appendix A](#))

[illegible]

## GetI2CErrorStats

Description: Returns the current I2C error statistics.

Syntax: `short GetI2CErrorStats(out string xmlErrorStats)`

Parameters: `xmlErrorStats` [out]current I2C error statistics returned in xml format

Return: Error code (see [Appendix A](#))

Sample: 

```
string xmlErrorStats = string.Empty;
short alarm = job.Device.GetI2CErrorStats(out xmlErrorStats);
```

## LoadCalibrationTableData

Description: Installs a calibration table's data.

Syntax:        short GetCalibrationTableData(string xmlCalibrationData)

Parameters:    xmlCalibrationData [in ] calibration table data in xml format

Return:        Error code (see [Appendix A](#))

Sample:

```
string xmlCalibrationData;//pre-defined table data
short alarm = job.Device.LoadCalibrationTableData(xmlCalibrationData);
```

## SetClock

Description: Assigns the printer's current system clock value.

```
Syntax:      short SetClock(
                byte      month,
                byte      day,
                int        year,
                byte      hour,
                byte      minute)
```

```
Parameters:  month      [in ]current month
             day        [in ]current day
             year       [in ]current year
             hour       [in ]current hour
             minute     [in ]current minute
```

Return: Error code (see [Appendix A](#))

Sample:

```
byte month = 1; //January
byte day = 20;
int year = 2014;
byte hour = 13; //24 hour format
byte minute = 10;
short alarm = job.Device.SetClock(month, day, year, hour, minute);
```



# Custom Mag Settings

This class is accessed through JobControl Namespace.

## Methods

GetMagTrackBitsPerChar .....	174
GetMagTrackDataParity .....	175
GetMagTrackDensity .....	176
GetMagTrackSentinelFormat .....	177
SetMagTrackBitsPerChar .....	178
SetMagTrackDataParity .....	179
SetMagTrackDensity .....	180
SetMagTrackSentinelFormat .....	181

## GetMagTrackBitsPerChar

Description: Returns the number of bits used per character for the specified magnetic track.

```
Syntax:      void GetMagTrackBitsPerChar(
                    MagTrackNumberEnum    trkNumber
                    out byte               bitsPerChar)
```

Parameters:	trkNumber	[in ]the magnetic track (see <a href="#">Appendix B</a> for details)
	bitsPerChar	[out]the number of bits per char for the magnetic track

```
Returns:      none
```

[illegible]

## GetMagTrackDataParity

Description: Returns the type of data parity in use for the magnetic track specified.

[illegible]

Parameters:	trkNumber	[in ]the magnetic track (see <a href="#">Appendix B</a> for details)
	dataParity	[out]the type of data parity in use(see <a href="#">Appendix B</a> for details)

```
Returns:      none
```

```
Sample:      //get the parity for magnetic track No 1:

MagTrackNumberEnum trkNumber = MagTrackNumberEnum.MagTrack1;
                                                    MagDataCharParityEnum
dataParity = MagDataCharParityEnum.Odd;

job.JobControl.CustomMagSettings.GetMagTrackDataParity(
            trkNumber out dataParity);
```

## GetMagTrackDensity

Description: Returns the density for the magnetic track specified.

```
Syntax: void GetMagTrackDensity(
                MagTrackNumberEnum   trkNumber
                out int               density);
```

Parameters:	trkNumber	[in ]the magnetic track (see <a href="#">Appendix B</a> for details)
	density	[out]the density of the track

Returns: none

```
Sample:      //get the track density for magnetic track No 1:

MagTrackNumberEnum trkNumber = MagTrackNumberEnum.MagTrack1;

int density = 0;

job.JobControl.CustomMagSettings.GetMagTrackDensity(
                                     trkNumber out density);
```

## GetMagTrackSentinelFormat

**Description:** Returns the current sentinel format and the current start and stop sentinels for the magnetic track specified.

**Syntax:**

```
void GetMagTrackSentinelFormat(
    MagTrackNumberEnum    trkNumber
    out MagDataFormatEnum  format
    out string             start
    out string             end);
```

**Parameters:**

trkNumber	[in ]the magnetic track (see <a href="#">Appendix B</a> for details)
format	[out]the sentinel format for the specified track (see <a href="#">Appendix B</a> for details)
start	[out]the start sentinel for the specified track
stop	[out]the end sentinel for the specified track

**Returns:** None

**Sample:**

```
//get the sentinel format start and stop sentinels for magnetic track No 1:

MagTrackNumberEnum trkNumber = MagTrackNumberEnum.MagTrack1;
MagDataFormatEnum format =
MagDataFormatEnum.Ascii;

string start = string.Empty;
string end =
string.Empty;
job.JobControl.CustomMagSettings.GetMagTrackSentinelFormat(
    trkNumber out format out start out end);
```

## SetMagTrackBitsPerChar

Description: Sets the number of bits used per character for the specified magnetic track.

[illegible]

Parameters:	trkNumber	[in ]the magnetic track (see <a href="#">Appendix B</a> for details)
	bitsPerChar	[in ]the number of bits per char for the magnetic track

Returns: none

```
Sample: //set the number of bits per character to 8 for magnetic
        track No 1:

        MagTrackNumberEnum trkNumber = MagTrackNumberEnum.MagTrack1;

        byte bitsPerChar = 8;

        job.JobControl.CustomMagSettings.SetMagTrackBitsPerChar(
            trkNumber bitsPerChar);
```

## SetMagTrackDataParity

Description: Sets the type of data parity in use for the magnetic track specified.

[illegible]

Parameters:	trkNumber	[in ]the magnetic track (see <a href="#">Appendix B</a> for details)
	dataParity	[in ]the type of data parity to use (see <a href="#">Appendix B</a> for details)

Returns: none

```
Sample:      //set odd parity for magnetic track No 1:

MagTrackNumberEnum trkNumber = MagTrackNumberEnum.MagTrack1;

MagDataCharParityEnum dataParity = MagDataCharParityEnum.Odd;

job.JobControl.CustomMagSettings.SetMagTrackDataParity(
            trkNumber dataParity);
```

## SetMagTrackDensity

Description: Sets the density for the magnetic track specified.

[illegible]

Parameters:	trkNumber	[in ]the magnetic track (see <a href="#">Appendix B</a> for details)
	density	[in ]the density of the track

Returns: none

```
Sample:      //set the track density for magnetic track No 1:

MagTrackNumberEnum trkNumber = MagTrackNumberEnum.MagTrack1;

int density = 8;

job.JobControl.CustomMagSettings.SetMagTrackDensity(
            trkNumber density);
```

## SetMagTrackSentinelFormat

**Description:** Sets the current sentinel format and the current start and stop sentinels for the magnetic track specified.

**Syntax:**

```
void SetMagTrackSentinelFormat(
    MagTrackNumberEnum    trkNumber
    MagDataFormatEnum     format
    string                 start
    string                 end);
```

**Parameters:**

trkNumber	[in ]the magnetic track (see <a href="#">Appendix B</a> for details)
format	[in ]the sentinel format for the specified track (see <a href="#">Appendix B</a> for details)
start	[in ]the start sentinel for the specified track
end	[in ]the end sentinel for the specified track

**Returns:** none

**Sample:** //set the sentinel format to ascii the start sentinel to < and the stop sentinel to > for magnetic track No 1:

```
MagTrackNumberEnum trkNumber = MagTrackNumberEnum.MagTrack1;

MagDataFormatEnum format = MagDataFormatEnum.Ascii;
string start = "<";
string end   = ">";

job.JobControl.CustomMagSettings.SetMagTrackSentinelFormat(
    trkNumber format start end);
```

# Laminator

## Methods

CalibrateLaminate .....	183
RestoreDefaultConfiguration .....	184
GetLaminateParams .....	186
GetLaminatorStatus .....	187
GetBottomLaminateOffsets .....	188
SetBottomLaminateOffsets .....	189
GetTopLaminateOffsets .....	190
SetTopLaminateOffsets .....	191
GetLaminationSpeedOffsets .....	192
SetLaminationSpeedOffsets .....	193
GetLaminatorOffsets .....	194
SetLaminatorOffsets .....	195
GetLaminatorSensorStates .....	196
GetLaminatorSensorValues .....	197
GetLaminatorCardCount .....	198
GetLaminatorInfo .....	199
LoadParameters .....	200
GetOdometerValues .....	201

## CalibrateLaminate

Description: Calibrates the laminate.

Syntax:        short CalibrateLaminate ( )

Parameters:   none

Returns:       error code (see Appendix )

Sample:       alarm = job.Laminator.CalibrateLaminate();

## RestoreDefaultConfiguration

Description: Resets the laminator to its default configuration.

Syntax:        short RestoreDefaultConfiguration ( )

Parameters:   none

Returns:       error code (see Appendix )

Sample:       alarm = job.Laminator.RestoreDefaultConfiguration();

## SaveParameters

Description: Saves the laminator's current configuration.

Syntax:        short SaveParameters()

Parameters:   none

Returns:       error code (see Appendix )

Sample:        alarm = job.Laminator.SaveParameters();

## GetLaminateParams

Description: Returns the laminate's parameters.

Syntax:        short GetLaminateParams (

	bool	topLaminate
	out int	type
	out string	partNumber
	out string	description
	out string	oem
	out int	initialSize
	out int	panelsRemaining )

Parameters:    topLaminate        [in ]true = return top laminate parameters  
     false = return bottom laminate parameters

type	[out]type of laminate
partNumber	[out]laminate part number
description	[out]description of laminate type
oem	[out]laminate OEM code
initialSize	[out]initial size of laminate
panelsRemaining	[out]number of panels remaining

Returns:        error code (see Appendix )

Sample:        alarm = job.Laminator.GetLaminateParams(topLaminate out type  
     out partNumber out description out oem out initialSize  
     out panelsRemaining);

## GetLaminatorStatus

Description: Returns the laminator's current status.

Syntax:           short GetLaminatorStatus (

out string	status
out int	errorCode
out int	jobsPending
out int	jobsActive
out int	jobsComplete
out int	jobErrors
out int	jobsTotal
out int	nextActionID )

Parameters:   status                   [out]laminator status

errorCode	[out]error code (see Appendix )
jobsPending	[out]number of pending jobs
jobsActive	[out]number of active jobs
jobsComplete	[out]number of completed jobs
jobErrors	[out]number of jobs with errors
jobsTotal	[out]total number of jobs
nextActionID	[out]action ID of next job

Returns:       error code (see Appendix )

Sample:       alarm = job.Laminator.GetLaminatorStatus(out status  
                    out errorCode out jobsPending out jobsActive  
                    out jobsComplete out jobErrors out jobsTotal  
                    out nextActionID);

## GetBottomLaminateOffsets

Description: Returns the bottom laminate's x offset and temperature offset.

[illegible]

Parameters:	xOffset	[out]x offset of bottom laminate
	temperatureOffset	[out]temperature offset of bottom laminate

Returns: error code (see Appendix )

```
Sample:      alarm = job.Laminator.GetBottomLaminateOffsets(out xOffset
              out temperatureOffset);
```

## SetBottomLaminateOffsets

Description: Sets the bottom laminate's x offset and temperature offset.

[illegible]

Parameters:	xOffset	[in ]x offset of bottom laminate
	temperatureOffset	[in ]temperature offset of bottom laminate

Returns: error code (see Appendix )

```
Sample:      alarm = job.Laminator.SetBottomLaminateOffsets(xOffset
               temperatureOffset);
```

## GetTopLaminateOffsets

Description: Returns the top laminate's x offset and temperature offset.

[illegible]

Parameters:	xOffset	[out]x offset of top laminate
	temperatureOffset	[out]temperature offset of top laminate

Returns: error code (see Appendix )

```
Sample:      alarm = job.Laminator.GetTopLaminateOffsets(out xOffset
              out temperatureOffset);
```

## SetTopLaminateOffsets

Description: Sets the top laminate's x offset and temperature offset.

[illegible]

Parameters:	xOffset	[in ]x offset of top laminate
	temperatureOffset	[in ]temperature offset of top laminate

Returns: error code (see Appendix )

```
Sample:      alarm = job.Laminator.SetTopLaminateOffsets(xOffset
               temperatureOffset);
```

## GetLaminationSpeedOffsets

Description: Returns the lamination speed offsets.

```
Syntax:      short GetLaminationSpeedOffsets (
                                out int          lamination
                                out int          passthru )
```

```
Parameters:  lamination      [out]lamination speed offset
             passthru        [out]pass through speed offset
```

Returns: error code (see Appendix )

```
Sample:      alarm = job.Laminator.GetLaminationSpeedOffsets(
              out lamination out passthru);
```

## SetLaminationSpeedOffsets

Description: Sets the lamination speed offsets.

```
Syntax:      short SetLaminationSpeedOffsets (
                                     int          lamination
                                     int          passthru )
```

```
Parameters:  lamination      [in ]lamination speed offset
              passthru       [in ]pass through speed offset
```

Returns: error code (see Appendix )

```
Sample:      alarm = job.Laminator.SetLaminationSpeedOffsets(
              lamination passthru);
```

## GetLaminatorOffsets

Description: Returns the patch length and coarse x offset.

[illegible]

Parameters:	patchLen	[out]patch length
	coarseXOffset	[out]coarse x offset

Returns: error code (see Appendix )

```
Sample:      alarm = job.Laminator.GetLaminatorOffsets(out patchLen
              out coarseXOffset);
```

## SetLaminatorOffsets

Description: Sets the patch length and coarse x offset.

```
Syntax:      short SetLaminatorOffsets (
                    int                patchLen
                    int                coarseXOffset )
```

Parameters:	patchLen	[in ]patch length
	coarseXOffset	[in ]coarse x offset

Returns: error code (see Appendix )

```
Sample:    alarm = job.Laminator.SetLaminatorOffsets(patchLen
            coarseXOffset);
```

## GetLaminatorSensorStates

Description: Returns the states of the laminator's sensors.

```
Syntax:      short GetLaminatorSensorStates(
                out object                      sensorStates )
```

Parameters:    sensorStates            [out]sensors' states

```
Examples:      Sensor & example state
              [0] "EntryBlocked: no"
              [1] "StagingBlocked: no"
              [2] "ExitBlocked: no"
              [3] "DoorOpen: no"
```

Returns: error code (see Appendix )

```
Sample:      object objSensorStates = null;

alarm = job.Laminator.GetLaminatorSensorStates(
            out objSensorStates );

if (objSensorStates != null)
{
    Array array = (Array)objSensorStates;
    string[] sensorStates = new string[array.GetLength(0)];
    for (int i = 0; i < array.GetLength(0); i++)
    {
        sensorStates[i] = (string)array.GetValue(i);
        if (i == 0) //EntryBlocked
            sensorState = sensorStates[i];
    }
}
```

## GetLaminatorSensorValues

Description: Returns the values of the laminator's sensors.

[illegible]

Parameters:    sensorValues            [out]sensors' values

Examples:      Sensor & example value

```
[0] "VoltageAC: 110"
[1] "TopMediaRaw: 1003"
[2] "BottomMediaRaw: 1004"
[3] "TopTemperature: 43"
[4] "BottomTemperature: 47"
```

Returns: error code (see Appendix )

```
Sample:      object objSensorValues = null;
```

```
alarm = job.Laminator.GetLaminatorSensorValues(
    out objSensorValues);

if (objSensorValues != null)
{
    Array array = (Array)objSensorValues;
    string[] sensorValues = new string[array.GetLength(0)];
    for (int i = 0; i < array.GetLength(0); i++)
    {
        sensorValues[i] = (string)array.GetValue(i);
        if (i == 0) //VoltageAC
            sensorValue = sensorValues[i];
    }
}
```

## GetLaminatorCardCount

Description: Returns the number of cards laminated and the number of cards processed.

[illegible]

```
Parameters:  cardsLaminated    [out]number of cards laminated
            cardsProcessed    [out]number of cards processed
```

Returns: error code (see Appendix )

```
Sample:      alarm = job.Laminator.GetLaminatorCardCount(out cardsLaminated
              out cardsProcessed);
```

## GetLaminatorInfo

**Description:** Returns the laminator's serial number firmware version and media version.

**Syntax:**

```
short GetLaminatorInfo (
                                out string      serialNumber
                                out string      firmwareVersion
                                out string      mediaVersion )
```

**Parameters:**

serialNumber	[out]laminator serial number
firmwareVersion	[out]laminator firmware version
mediaVersion	[out]laminator media version

**Returns:** error code (see Appendix )

**Sample:**

```
alarm = job.Laminator.GetLaminatorInfo(out serialNumber
                                         out firmwareVersion out mediaVersion);
```

## LoadParameters

Description: Loads the laminator's current configuration into RAM.

Syntax:        short LoadParameters()

Parameters:    none

Returns:       error code (see Appendix )

Sample:        alarm = job.laminator.LoadParameters();

## GetOdometerValues

Description: Returns the values of the laminator's odometers.

Syntax:        short GetOdometerValues(out object odometerValues)

Parameters:   odometerValues                      [out] odometer values

Examples:      Odometer example values - Times measured in seconds

```
[0] "TotalUpTime: 212824"
[1] "TopBulbOnTime: 8835"
[2] "BottomBulbOnTime: 8584"
```

Returns:        error code (see Appendix )

```
Sample:        object objOdometerValues = null;

alarm = job.Laminator.GetOdometerValues(out objOdometerValues);

if (objOdometerValues!= null) {
    Array array = (Array)
objOdometerValues;
    string[] odometerValues = new string[array.GetLength(0)];
    for (int i = 0; i < array.GetLength(0); i++)
    {
        odometerValues[i] =
        (string)array.GetValue(i);
    }
}
```

# Utilities

## Methods

ByteArrayToVariantArray .....	203
BytePtrToVariantArray .....	204
IntArrayToVariantArray .....	205
LongArrayToVariantArray .....	206
VariantArrayToByteArray .....	207
VariantArrayToIntArray .....	208
VariantArrayToLongArray .....	209

## ByteArrayToVariantArray

Description: Creates a variant array type from a byte array type.

Syntax:       void job.Utilities.ByteArrayToVariantArray (

object	byteArray
out object	variantArray )

Parameters:   byteArray           [in ]byte array to be converted

variantArray	[out]variant array created from byte array
--------------	--

Returns:       nothing

Sample:       job.Utilities.ByteArrayToVariantArray(byteArray

out variantArray);
--------------------

## BytePtrToVariantArray

Description: Creates a variant array type from a byte pointer type.

Syntax:           object job.Utilities.BytePtrToVariantArray (

	ref byte	bytePtr
	int	byteCount )

Parameters:   bytePtr                   [in ]pointer to byte array

byteCount	[in ]number of bytes in the byte array
-----------	--

Returns:       variantArray created

Sample:       object variantArray = job.Utilities.BytePtrToVariantArray

(byteArray byteCount);
------------------------

## IntArrayToVariantArray

Description: Creates a variant array type from an integer array type.

Syntax:       void job.Utilities.IntArrayToVariantArray (

object	intArray
out object	variantArray )

Parameters:   intArray               [in ]integer array to be converted

variantArray	[out]variant array created from integer array
--------------	---

Returns:       nothing

Sample:       job.Utilities.IntArrayToVariantArray(intArray

out variantArray);
--------------------

## LongArrayToVariantArray

Description: Creates a variant array type from a long array type.

Syntax:       void job.Utilities.LongArrayToVariantArray (

object	longArray
out object	variantArray )

Parameters:   longArray           [in ]long array to be converted

variantArray	[out]variant array created from long array
--------------	--

Returns:       nothing

Sample:       job.Utilities.LongArrayToVariantArray(longArray

out variantArray);
--------------------

## VariantArrayToByteArray

Description: Creates a byte array type from a variant array type.

Syntax:       void job.Utilities.VariantArrayToByteArray (

	object	variantArray
	out object	byteArray )

Parameters:   variantArray       [in ]variant array to be converted

byteArray	[out]byte array created from variant array
-----------	--

Returns:       nothing

Sample:       job.Utilities.VariantArrayToByteArray (variantArray

out   byteArray);
-------------------

## VariantArrayToIntArray

Description: Creates an integer array type from a variant array type.

Syntax:       void job.Utilities.VariantArrayToIntArray (

	object	variantArray
	out object	intArray )

Parameters:   variantArray       [in ]variant array to be converted

intArray	[out]integer array created from variant array
----------	---

Returns:       nothing

Sample:       job.Utilities.VariantArrayToIntArray (variantArray

out intArray)
---------------

## VariantArrayToLongArray

Description: Creates a long array type from a variant array type.

Syntax:       void job.Utilities.VariantArrayToLongArray (

	object	variantArray
	out object	longArray )

Parameters:   variantArray       [in ]variant array to be converted

longArray	[out]long array created from variant array
-----------	--

Returns:       nothing

Sample:       job.Utilities.VariantArrayToLongArray (variantArray

out longArray);
-----------------

# Appendix A

## Error Codes

### Introduction

This appendix lists error codes error messages and descriptions for all error messages that may appear when running applications created with the SDK for Zebra Card Printers.

## Errors and Alarms

### Errors

Errors are thrown exceptions generated by an SDK function. Errors are captured by the try catch syntax.

```
try
{
    short alarmValue = SDK functions
}
catch (COMException comEx)    // to capture COM exceptions
{
    int comErr = comEx.ErrorCode & 0xff;
}
catch (Exception ex)          // to capture other function exception
{
    string exMessage = ex.Message;
}
```

### Alarms

Alarms are generated by a ZMotif device and captured by the SDK functions. They are typically mechanical card movement ribbon and film alerts. Alarms are independent of ZMotif jobs and indicate if it is safe to proceed to the next job. They are returned as numbers.

```
short alarmValue = SDK Funtion ( ... )

if ( alarmValue not equal 0 )
    errMsg = job.Device.GetStatusMessageString(alarmValue);
else
    proceed to next job
```

## Error Codes and Descriptions

CODE	DESCRIPTION
00000 - 00999 system errors or events	
00000	No error
00001	Printer powering up
00002	Boot region integrity error
00003	Program region integrity error
00004	Watchdog timer error
00005	Incompatible firmware upgrade attempted
00006	EP diagnostic mode error
00007	Firmware upgrade failed
00008	Critical error
01000 - 01999 ZMJ errors	
01001	Invalid command
01002	Command processing error
01003	Job Control XML parse error
01004	Job already open
01005	Invalid job ID
01006	Invalid ZMotif version
01007	Number of requested copies out-of-range
01008	Command identifier 'ZBR1' not found
01009	No XML data received
01010	No job type received
01011	Unknown job type received
01012	Data decryption error
01013	No magnetic encoder installed
02000 - 02999 imaging errors	
02001	Image to print area error
02002	Print to media area error
02003	Font render error
02004	Drawing render error
02005	Invalid image processing data
02006	Error receiving IPD
02007	Error sending IPD to job scheduler
02008	Received incomplete image data
02009	Image processing aborted
03000 - 03999 host & communication errors, mostly ZMC	
03001	Printer offline
03002	Printer busy

## Appendix A Error Codes

CODE	DESCRIPTION
03003	Invalid ZMC command
03004	Invalid ZMC sub-command
03005	Invalid ZMC parameter (1)
03006	Invalid ZMC parameter (2)
03007	Invalid ZMC parameter (3)
03008	Command processing error
03009	Response too large for host
03010	Host write occurred when host read expected
03011	Host read occurred when host write expected
03012	Data less than specified in header
03013	Data more than specified in header
03014	Communication synch error
03015	End Action error
03016	Cancel Action error
03017	No Start Action
03018	Start Action already called
03019	Job data error
03020	Memory-pool allocation error
03021	XML parse error
03022	Invalid payload length
03023	HMAC missing
03024	Invalid payload content
03025	Device reservation failed
04000 - 04999 media errors (card, laminate, retransfer film, paper, etc)	
04001	Out of cards
04002	Invalid card type
04003	Card jam
04004	Reserved
04005	Reserved
04006	Reserved
04007	Reserved
04008	Reserved
04009	Reserved
04010	Out of retransfer media
04011	Invalid retransfer media
04012	Retransfer media jam
04013	Retransfer media motion error
04014	Card not detected
04015	Insert card timeout
04016	Card feeder is empty
04017	Invalid retransfer media

## Appendix A Error Codes

CODE	DESCRIPTION
05000 - 05999 donor errors (ribbon)	
05001	Out of ribbon
05002	Invalid ribbon
05003	Ribbon jam
05004	Ribbon motion error
05005	Ribbon ADC error
05006	Ribbon BEMF error
05007	Ribbon color detection error
05008	Invalid ribbon
06000 - 06999 memory/storage errors (RAM, NVMEM, external flash drive, etc)	
06001	Out of RAM
06002	Out of external flash
06003	Out of internal flash
06004	Out of NVM
06005	Data store error
06006	Data delete error
06007	Font store error
06008	Font delete error
06009	Program FPGA failure
06010	Erase FPGA failure
06011	Program EP failure
06012	Erase EP failure
06013	Program IP failure
06014	Erase IP failure
06015	Pool allocation error
06016	Pool de-allocation error
06017	NVM EP communication error
06018	NVM CRC error
06019	NVM access error
06020	NVM initialization error
06021	NVM backup error
06022	NVM restore error
06023	NVM open or close error
06024	NVM program backup error
06025	NVM erase backup error
07000 - 07999 engine errors (feed, flip, head, doors, etc.)	
07001	Card feed error
07002	Card cleaning error
07003	Printhead cable disconnected
07004	Card eject error
07005	Printhead temperature too high

## Appendix A Error Codes

CODE	DESCRIPTION
07006	Printhead temperature too low
07007	Protocol error
07008	Door open
07009	Invalid EP script
07010	Printhead CAM home error
07011	Transfer rollers temperature too high
07012	Transfer rollers temperature too low
07013	Motor voltage range error
07014	EP script processing error
07015	Mag retrace error
07016	Card transfer error
07017	Card reject error
07018	SmartCard positioning error
07019	EP script content error
07020	EP script transmission error
07021	Print path initialization error
07022	Flipper initialization error
07023	SmartCard cam error
07024	Options module card jam
07025	Print path card jam
07026	Flipper card jam
07027	Media drawer open
07028	Feeder door open
07029	Flipper move failure
07030	Reserved
07031	Reserved
07032	ATM card jam
07033	Reserved
07034	Reject bin full
07035	Magnetic encoder card jam
07036	Print path card jam
07037	Print synchronization card jam
07038	Print entrance card jam
07039	Print exit card jam
07040	Flipper initialization error
07041	Printhead initialization error
07042	Print path initialization error
07043	Options module initialization error
07044	SmartCard initialization error
08000 - 08999 OCP errors	
08001	Unresponsive
08002	Transmit error

## Appendix A Error Codes

CODE	DESCRIPTION
09000 - 09999 encoding errors, magnetic stripe	
09001	Read error
09002	Write verification error
09003	Receive error
09004	No magnetic stripe detected
10000 - 10999 encoding errors, smartcard contact	
10001	Read error
10002	Write verification error
11000 - 11999 encoding errors, smartcard contactless	
11001	Read error
11002	Write verification error
12000 - 12999 USB errors	
12001	Locked
12002	Open failed
12003	Handle error
12004	Message short
12005	Message error
12006	Payload pending
12007	Payload too big
12008	Restart
12009	Synchronization error
13000 - 13999 job management and processing errors	
13001	Create job error
13002	Queue error
13003	Action ID not found
13004	Insufficient memory available to accept job
13005	EP Processing error
13006	Job cancelled
13007	Job aborted
13008	Job buffer full
14000 - 14999 halogen control board errors	
14001	Control board missing
14002	Bulb error
14003	Sensor error
14004	Bootloader mode - firmware reload may be required

## Appendix A Error Codes

CODE	DESCRIPTION
15000 - 15999 media authentication board (MAB) errors	
15001	Control board missing
15002	Bootloader mode - firmware reload may be required
16000 - 16999 security-related errors	
16001	Invalid passkey
16002	Invalid crypto key
16003	Authentication failed
16004	Invalid printer data
16005	Invalid HMAC
16006	Unsupported action
17000 - 17499 laminator board faults	
17001	Missing laminator
17002	Initialization error
17003	Unknown error
17004	Media authentication board missing
17005	Top laminate feed error
17006	Bottom laminate feed error
17007	Top laminate registration error
17008	Staging error
17009	Early card jam
17010	Mid card jam
17011	Late card jam
17012	Poll timeout
17013	Top heater error - power off printer and correct problem
17014	Bottom heater error - power off printer and correct problem
17015	Top heater over temperature - power off printer and correct problem
17016	Bottom heater over temperature - power off printer and correct problem
17017	Top cutter stall - power off printer and correct problem
17018	Bottom cutter stall - power off printer and correct problem
17019	Top cutter failure - power off printer and correct problem
17020	Bottom cutter failure - power off printer and correct problem
17021	Top sensor failure - Power off printer and correct problem
17022	Bottom sensor failure - Power off printer and correct problem
17023	Fan failure - Power off printer and correct problem
17024	EEPROM corrupt
17025	Reserved
17026	Out of top and bottom laminate
17027	Out of top laminate
17028	Out of bottom laminate
17029	Invalid top laminate
17030	Invalid bottom laminate

## Appendix A Error Codes

CODE	DESCRIPTION
17031	Bottom laminate registration error
17032	Remove top laminate
17033	Remove bottom laminate
17034	Remove top and bottom laminate
17035	Reserved
17036	Reserved
17037	Reserved
17038	Door open
17039	Reserved
17040	Initializing
17041	Bootloader mode - firmware reload may be required
17042	MAB Bootloader mode - firmware reload may be required
18000 - 18999 wired network (ethernet) errors	
18001	Open failed
19000 - 19999 wireless network (WiFi) errors	
19001	Open failed
19002	Access point missing
19003	Link lost
19004	Incompatible configuration
19005	Association failed
19006	Connection failed
65000 SDK+ errors	
65001	Device not open
65002	Device already open
65003	Device not available
65004	Device not responding
65005	Bad ZMC response signature
65006	Bad ZMC Command echo
65007	Insufficient data received from device
65008	Invalid argument
65009	Path to XML element not found
65010	Parse error
65011	Empty/Invalid Data Structure
65012	Buffer overflow
65013	SmartCard Encoder not available
65014	Encryption error
65015	Job status error
65016	Dual sided printing not supported
65017	Unable to obtain exclusive access to device
65018	Device in session with another host

## Appendix A Error Codes

CODE	DESCRIPTION
65019	Invalid device for requested operation
65020	Passphrase or security key required for requested operation
65021	Memory allocation error
65022	No devices found
65023	Disconnect error
65024	WiFi not available
65025	Invalid media for requested operation
65026	Requested operation timed out

# Appendix B

# SDK Enumerations

## Graphic Enums

FontTypeEnum

Bold

Italic

Regular

Strikeout

Underline

ImageOrientationEnum

Landscape

Portrait

ImagePositionEnum

Centered

LowerLeft

LowerRight

UpperLeft

UpperRight

MonochromeConversion

Diffusion

HalfTone\_6x6

HalfTone\_8x8

None

NetworkIPVersion

IPv4

IPv6

### PrinterModelTypeEnum

- ZXPSeries3
- ZXPSeries7
- ZXPSeries8
- ZXPSeries9
- ZXPSeries9\_600

### RibbonTypeEnum

- Color
- GrayDye
- GrayUV
- Inhibit
- Monok\_NoPanels
- MonoK
- MonoUV
- Overlay
- GrayG
- MonoG

### RotationTypeEnum

- Rotate180FlipXY
- RotateNoneFlipNone
- Rotate270FlipXY
- Rotate90FlipNone
- Rotate180FlipNone
- RotateNoneFlipXY
- Rotate270FlipNone
- Rotate90FlipXY
- Rotate180FlipY
- RotateNoneFlipX
- Rotate90FlipX
- Rotate270FlipY
- RotateNoneFlipY
- Rotate180FlipX
- Rotate90FlipY
- Rotate270FlipX

### TextAlignmentEnum

Near

Center

Far

### TextRenderingEnum

AntiAlias

AnitAliasGridFit

ClearTypeGridFit

SingleBitPerPixel

SingleBitPerPixelGridFit

## Job Enums

### BoolTypeEnum

False\_BT

True\_BT

NoChange

### CalibrationRoutineEnum

AutoSetDefaults

CardCenterPosition

CardToHeaterRoller

CardToTransfer

CardTransferToCenter

ContactPosition

MagPosition

RibbonDamping

RibbonTorque

Tricolor

### CalibrationTableEnum

LUT1Default

LUT1Installed

LUT2Default

LUT2Installed

LUTsDefault

LUTsInstalled

### CalibrationTypeEnum

Calc

HighTorque

LowTorque

### CapabilitiesReportTypeEnum

General

MediaType

### CleanPathTypeEnum

FeederPath

ATMPath

TransferPath

### ComponentTypeEnum

- ContactProbe
- PrintHead
- TransferRollers

### ConnectionTypeEnum

- All
- Ethernet
- USB

### DataSourceEnum

- NoData
- Track1Data
- Track2Data
- Track3Data

### DestinationTypeEnum

- Eject
- Feeder
- Reject

### EMailAttachmentsEnum

- Configuration
- ErrorLog
- EventsLog
- GeneralStatus
- NoAttachments

### EncryptionTypeEnum

- AES
- NoEncryption

### ErrorControlLevelEnum

- EC\_High
- EC\_Medium
- EC\_None

### EventLogTypeEnum

- AlarmEvents
- ErrorAlarmEvents
- ErrorEvents
- NoEvents

### FanTypeEnum

- PrintheadFan
- TransferCardFan
- TransferRollerFan

### FeederSourceEnum

- ATMSlot
- CardFeeder

### GraphicTypeEnum

- BluePlane
- BMP
- GrayPlane
- GreenPlane
- MonoPlane
- NA
- RedPlane

### LogTypeEnum

- CleanEvents
- Errors
- Events
- Service

### MagCoercivityEnum

- HighCo
- LowCo

### MagDataCharParityEnum

- Even
- Odd

### MagDataFormatEnum

- ASCII
- Hex

## MagEncodingTypeEnum

AAMVA

Ballys

ISO

JIS

VingCard

## MagTrackNumberEnum

MagTrack1

MagTrack2

MagTrack3

## MediaTypeEnum

ColorRibbon

## MemorySourceEnum

MemFlash

MemExternal

MemRAM

## MonoConvTypeEnum

MonoBarcode

MonoDiffusion

MonoHalftone

MonoText

## NetworkIPVersion

IPv4

IPv6

## OrientationEnum

Landscape

Portrait

## PanelTypeEnum

Black

Cyan

Magenta

UV

Yellow

## PrintOptimizationModeEnum

Quality

Speed

## PrintTypeEnum

Color  
GrayDye  
GrayUV  
Inhibit  
Monok\_NoPanels  
MonoK  
MonoUV  
Overlay  
WhiteResin  
Helper  
GrayG  
MonoG

## ReservationTypeEnum

PendingSession  
ImmediateSession

## ResetTypeEnum

Warm  
Full

## RFFieldTypeEnum

RFContactless  
RFLaminate  
RFRibbon

## RotationEnum

Rotate\_0  
Rotate\_180

## SharpnessLevelEnum

Off  
Low  
Normal  
High

## SideEnum

Back  
Front

## SmartCardCommChannelEnum

SC\_Ethernet  
SC\_USB

### SmartCardTypeEnum

Contact

iClass

FeliCa

Legic

MIFARE

None

Prox

UHF

### StandbyTimeoutEnum

FourHours

Never

OneHour

ThirtyMinutes

TwoHours

### SystemTypeEnum

CardTransport

PrinterMechanism

RetransferFilm

### TransferTypeEnum

dualSided

SingleSided

### TransportTypeEnum

ATMToCenter

CamToX

CamToY

CenterToOutput

CenterToReject

CenterToTransfer

HopperToCenter

HopperToContact

HopperToContactless

### USBSpeedEnum

FullSpeed

HighSpeed

### WirelessCryptoTypeEnum

CCMP

NoCrypto

RC4

TKIP

### WirelessSecurityTypeEnum

Open

WEP104

WEP40

WPA

WPA2

# Appendix C

## XML Documents

### Capabilities

```
<?xml version="1.0" encoding="UTF-8" ?>
<capabilities>
  <!--Printer-to-device ports printer acting as host e.g. USB (host mode) for
  external flash drive; Serial for internal smartcard encoder modules-->
  <host_ports>
    <host_port>
      <port_id>0</port_id>
      <type>usb_2_0</type>
    </host_port>

    <host_port>
      <port_id>1</port_id>
      <type>usb_2_0</type>
    </host_port>

    <host_port>
      <port_id>2</port_id>
      <type>usb_2_0</type>
    </host_port>

    <host_port>
      <port_id>3</port_id>
      <type>usb_2_0</type>
    </host_port>

    <host_port>
      <port_id>4</port_id>
      <type>serial</type>
    </host_port>
  </host_ports>

  <!--Host-to-printer ports printer acting as device-->
  <device_ports>
    <device_port>
      <port_id>0</port_id>
      <type>usb_2_0</type>
    </device_port>

    <device_port>
      <port_id>1</port_id>
      <type>ethernet_10_100_1G</type>
    </device_port>

    <device_port>
      <port_id>2</port_id>
      <type>802.11_b_g</type>
    </device_port>
  </device_ports>
</capabilities>
```

```

        </device_port>
    </device_ports>

    <!--Software interfaces used by host applications-->
    <software_interfaces>
        <software_interface>
            <protocol>snmp</protocol>
            <transport>udp</transport>
            <port>161</port>
        </software_interface>

        <software_interface>
            <protocol>snmp_trap</protocol>
            <transport>udp</transport>
            <port>162</port>
        </software_interface>

        <software_interface>
            <protocol>http</protocol>
            <transport>tcp</transport>
            <port>80</port>
        </software_interface>

        <software_interface>
            <protocol>zmotif</protocol>
            <transport>tcp</transport>
            <port>9100</port>
        </software_interface>

        <software_interface>
            <protocol>zmotif</protocol>
            <transport>tcp</transport>
            <port>9099</port>
        </software_interface>

        <software_interface>
            <protocol>zmotif</protocol>
            <transport>usb_2_0</transport>
            <port></port>
        </software_interface>
    </software_interfaces>

    <!--This section specifies the supported languages for the Operator
    Control Panel.-->
    <ocp_languages>
        <ocp_language>
            <type>0</type>
            <name>English</name>
        </ocp_language>

        <ocp_language>
            <type>1</type>
            <name>French</name>
        </ocp_language>
    </ocp_languages>

```

```

    <ocp_language>
      <type>2</type>
      <name>Spanish</name>
    </ocp_language>

    <ocp_language>
      <type>3</type>
      <name>German</name>
    </ocp_language>

    <ocp_language>
      <type>4</type>
      <name>Portuguese</name>
    </ocp_language>

    <ocp_language>
      <type>5</type>
      <name>Italian</name>
    </ocp_language>
  </ocp_languages>

  <!--This section specifies the supported image sharpening levels.-->
  <image_sharpening_levels>
    <image_sharpening_level>off</image_sharpening_level>
    <image_sharpening_level>normal</image_sharpening_level>
    <image_sharpening_level>high</image_sharpening_level>
  </image_sharpening_levels>

  <!--Passkey protection implies host authentication and data encryption.-->
  <security>
    <passkey_protection>no</passkey_protection>
    <encryption_algorithm>none</encryption_algorithm>
  </security>

  <!--All memory values specified in kilobytes.-->
  <memory>
    <ram>65536</ram>
    <flash>16384</flash>
  </memory>

  <!--Supported graphic formats for images.-->
  <graphic_formats>
    <graphic_format>bmp</graphic_format>
    <graphic_format>raw</graphic_format>
  </graphic_formats>

  <!--List of all stored templates graphics text and barcode fonts available
  including temporary downloads-->
  <fonts>
    <font>
      <name>dejavu</name>
      <scalable>yes</scalable>
      <style>regular</style>
    </font>

    <font>
      <name>dejavu</name>
      <scalable>yes</scalable>
      <style>bold</style>
    </font>
  </font>

```

```

        <font>
            <name>dejavu</name>
            <scalable>yes</scalable>
            <style>italic</style>
        </font>
    </fonts>

    <templates>
        <template>
            <name>usb0:\templates\tplt_01.xml</name>
        </template>

        <template>
            <name>usb0:\templates\tplt_02.xml</name>
        </template>
    </templates>

    <graphics>
        <graphic>
            <name>usb0:\images\corplogo.bmp</name>
        </graphic>

        <graphic>
            <name>usb0:\images\visitext.bmp</name>
        </graphic>

        <graphic>
            <location>flash</location>
            <name>iffound.bmp</name>
        </graphic>
    </graphics>

    <print_system>retransfer</print_system>
    <print_type>2_side</print_type>

    <!--ATM-style card input refers to a single card bypass feed.-->
    <media_path>
        <atm>yes</atm>
        <magazine_capacity>100</magazine_capacity>
    </media_path>

    <!--Image size (in pixels) and resolution (in pixels per inch) information.-->
    <image>
        <logical_page>
            <length>1024</length>
            <width>648</width>
        </logical_page>

        <physical_page>
            <length>1012</length>
            <width>638</width>
        </physical_page>

        <print_resolution>300</print_resolution>
    </image>

    <!--The following mag section specifies only the physical installation. Formatting
    is defined at the ZMJ job level. Mag head types are ISO & JIS possibly others.-->
    <mag_encoder>iso</mag_encoder>
    <mag_stripe>back</mag_stripe>

```

```

<!--The following smart card encoding section specifies only the physical
installation of the internal encoding module.-->
<internal_encoder>
    <comms_channel>usb_2_0</comms_channel>
    <contact_encoding>yes</contact_encoding>
    <contactless_encoder>none</contactless_encoder>
</internal_encoder>

<!--List of all barcodes that can be natively rendered by the printer.-->
<barcodes>
    <barcode>code_39</barcode>
    <barcode>code_128b</barcode>
    <barcode>code_128c</barcode>
    <barcode>standard_2of5</barcode>
    <barcode>interleaved_2of5</barcode>
    <barcode>upc_a</barcode>
    <barcode>ean_8</barcode>
    <barcode>ean_13</barcode>
</barcodes>
</capabilities>

```

## Card Configurations

### Example 1

```

<configuration>
<cards>
<card>
  <!--General card information for custom card types.-->
  <!--All card thickness values are specified in mils.-->
  <information>
    <type>0</type>
    <zebra_part_number>Custom 1</zebra_part_number>
    <description>Custom 1</description>
    <thickness>30</thickness>
    <mag_stripe>no</mag_stripe>
    <coercivity/>
    <contact_encode>no</contact_encode>
    <non_contact_encode>no</non_contact_encode>
  </information>
  <!--Temperature settings for single-sided and double-sided transfers.-->
  <!--All temperatures and offsets are specified in Celsius.-->
  <transfer_temps>
    <top_single min="0" max="220">185</top_single>
    <top_double min="0" max="220">175</top_double>
    <bot_double min="0" max="220">165</bot_double>
  </transfer_temps>
  <!--Input and output speed settings for single-sided and double-sided transfers.-->
  <!--All transfer speeds are specified in inches per second.-->
  <transfer_speeds>
    <input_single min="1.00" max="8.00">1.50</input_single>
    <output_single min="1.00" max="8.00">1.75</output_single>
    <input_double min="1.00" max="8.00">1.50</input_double>
    <output_double min="1.00" max="8.00">1.75</output_double>
  </transfer_speeds>
  <!--Calibration table associated with the card type.-->
  <cal_table min="1" max="2">1</cal_table>
</card>
</cards>
</configuration>

```

## Example 2

```

<configuration>
<cards>
<card>
  <!--General card information for custom card types.-->
  <!--All card thickness values are specified in mils.-->
  <information>
    <type>1</type>
    <zebra_part_number>Custom 2</zebra_part_number>
    <description>Custom 2</description>
    <thickness>30</thickness>
    <mag_stripe>no</mag_stripe>
    <coercivity/>
    <contact_encode>no</contact_encode>
    <non_contact_encode>no</non_contact_encode>
  </information>
  <!--Temperature settings for single-sided and double-sided transfers.-->
  <!--All temperatures and offsets are specified in Celsius.-->
  <transfer_temps>
    <top_single min="0" max="220">185</top_single>
    <top_double min="0" max="220">175</top_double>
    <bot_double min="0" max="220">165</bot_double>
  </transfer_temps>
  <!--Input and output speed settings for single-sided and double-sided transfers.-->
  <!--All transfer speeds are specified in inches per second.-->
  <transfer_speeds>
    <input_single min="1.00" max="8.00">1.50</input_single>
    <output_single min="1.00" max="8.00">1.75</output_single>
    <input_double min="1.00" max="8.00">1.50</input_double>
    <output_double min="1.00" max="8.00">1.75</output_double>
  </transfer_speeds>
  <!--Calibration table associated with the card type.-->
  <cal_table min="1" max="2">1</cal_table>
</card>
</cards>
</configuration>

```

## Configuration

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>
  <!--Ethernet MAC address-->
  <physical_address>00:00:00:00:00:00</physical_address>

  <!--Current wired Ethernet settings-->
  <ethernet>
    <dhcp>enabled</dhcp>
    <ip_address>0.0.0.0</ip_address>
    <subnet_mask>0.0.0.0</subnet_mask>
    <gateway>0.0.0.0</gateway>
  </ethernet>

  <!--Current OCP language selection-->
  <ocp_language>
    <type>0</type>
    <name>English</name>
  </ocp_language>

  <!--Current usage counter values since last reset-->
  <usage_counters>
    <num_cards_printed>1526</num_cards_printed>
    <num_lines_printed>946728</num_lines_printed>
  </usage_counters>

  <!--Current standby timer setting in minutes.-->
  <!--A values of 0 indicates that the printer will never enter standby.-->
  <!--Values from 1 to 30 result in a timeout of 30 minutes.-->
  <!--Values from 31 to 60 result in a timeout of 60 minutes or 1 hour.-->
  <!--Values from 61 to 120 result in a timeout of 120 minutes or 2 hours.-->
  <!--Values greater than 120 result in a timeout of 480 minutes or 8 hours.-->
  <standby_timeout>60</standby_timeout>

  <!--User-settable imaging parameters-->
  <imaging_parameters>
    <!--Printhead resistance in ohms and color panel power adjustments.-->
    <printhead_resistance min="2975" max="4025">3500</printhead_resistance>

    <!--Manufacturing power adjustments are percentages of the pre-determined
    fixed energy value assigned to each color panel.-->
    <!--As the value goes up the density of that color panel will increase; and
    as the value goes down the density of that color panel will decrease.-->
    <!--If relative power levels for each color panel are adjusted individually
    by differing amounts the color balance of full color images will change.-->
    <mfg_power_adjustments>
      <yellow min="50" max="150">100</yellow>
      <magenta min="50" max="150">100</magenta>
      <cyan min="50" max="150">100</cyan>
      <k-resin_front min="50" max="150">100</k-resin_front>
      <k-resin_back min="50" max="150">100</k-resin_back>
      <k-dye min="50" max="150">100</k-dye>
      <overlay min="50" max="150">100</overlay>
      <uv_front min="50" max="150">100</uv_front>
      <uv_back min="50" max="150">100</uv_back>
    </mfg_power_adjustments>
  </imaging_parameters>
</configuration>
```

```

    <!--User power adjustments are percentages of the values established by
    the above manufacturing power adjustments.-->
    <user_power_adjustments>
        <yellow min="90" max="110">100</yellow>
        <magenta min="90" max="110">100</magenta>
        <cyan min="90" max="110">100</cyan>
        <k-resin_front min="90" max="110">100</k-resin_front>
        <k-resin_back min="90" max="110">100</k-resin_back>
        <k-dye min="90" max="110">100</k-dye>
        <overlay min="90" max="110">100</overlay>
        <uv_front min="90" max="110">100</uv_front>
        <uv_back min="90" max="110">100</uv_back>
    </user_power_adjustments>
</imaging_parameters>

<!--Current tri-color sensor values as determined by tri-color sensor calibration-->
<!--These values are not directly user-settable.-->
<tricolor_settings>
    <red>-28</red>
    <green>-47</green>
    <blue>0</blue>
</tricolor_settings>

<!--Current DC ribbon motor settings-->
<!--Units for each setting are as follows:-->
<!--    S0 in RPM-->
<!--    K0 in mA settable by user or via ribbon motor torque calibration-->
<!--    Kb in mA-min2/rev2-->
<!--    Ka in mA-min/rev-->
<!--    M0 in mA/in-lb settable by user or via ribbon motor torque calibration-->
<!--    Mb in mA-min2/in-lb-2/rev2-->
<!--    Ma in mA-min/in-lb-2/rev-->
<!--    VtoI in mA/V not user-settable-->
<ribbon_motor_parameters>
    <takeup_motor>
        <S0 min="5.00 " max="50.00">33.3000</S0>
        <K0 min="30.00" max="150.00">71.6600</K0>
        <Kb min="-1.50" max="1.50 ">0.5080</Kb>
        <Ka min="-0.10" max="0.10 ">-0.0077</Ka>
        <M0 min="50.00" max="300.00">171.5200</M0>
        <Mb min="-1.00" max="1.00 ">0.1750</Mb>
        <Ma min="-0.10" max="0.10 ">-0.0049</Ma>
        <VtoI>66.80</VtoI>
    </takeup_motor>
</ribbon_motor_parameters>

<!--Current ribbon registration setting in millimeters-->
<ribbon_sensor_offset min="1" max="12">6</ribbon_sensor_offset>

<!--Current back EMF potentiometer setting as determined by ribbon damping
calibration. Provides the analog voltage that maps to the speed of the motor.-->
<!--Valid range of values from 2 to 120. This value is not directly user-settable.-->
<bemf_potentiometer>61</bemf_potentiometer>

<!--Mechanical adjustment parameters-->
<!--All parameters in mils except film_print_x_offset which is in dots.-->
<mech_adjustments>
    <!--Steps from sensor to print in X and Y direction-->
    <film_print_x_offset min="-50" max="50">3</film_print_x_offset>
    <film_print_y_offset min="-100" max="100">0</film_print_y_offset>

```

```

<!--Steps from sensor to transfer-->
<film_transfer_start_offset
  min="0" max="200">100
</film_transfer_start_offset>

<!--Steps from the card detect sensor to the card in center of card the
transport in the X direction-->
<card_center_x_offset min="-150" max="150">55</card_center_x_offset>

<!--Steps from the card detect sensor to the card at the mag encode start
in the X direction-->
<card_mag_x_offset min="-150" max="150">-60</card_mag_x_offset>

<!--Steps from the card detect sensor to the card at the smart card encode
in the X direction-->
<card_smart_card_x_offset min="-300" max="300">0</card_smart_card_x_offset>

<!--Steps from the card detect sensor to the card in center of the transport
in the Y direction-->
<card_center_y_offset min="-300" max="300">-160</card_center_y_offset>

<!--Steps from card detect to transfer start in Y direction-->
<card_transfer_start_y_offset
  min="0" max="300">143
</card_transfer_start_y_offset>

<!--Steps from card detect to transfer end in Y direction-->
<card_transfer_end_y_offset
  min="0" max="1000">283
</card_transfer_end_y_offset>

<!--Steps from transfer start to transfer roller in Y direction-->
<card_transfer_roll_y_offset
  min="0" max="1000">333
</card_transfer_roll_y_offset>
</mech_adjustments>

<!--Media information as read from the RFID tags resident on the media rolls-->
<media_info>
  <!--Ribbon information-->
  <ribbon>
    <type>101</type>
    <zebra_part_number>800133-480</zebra_part_number>
    <description>YMCKK</description>
    <oem_country>ELTR</oem_country>
    <initial_size>510</initial_size>
    <panels_remaining>393</panels_remaining>
  </ribbon>

  <!--Film information-->
  <film>
    <type>104</type>
    <zebra_part_number>800133-600</zebra_part_number>
    <description>Standard</description>
    <oem_country>ELTR</oem_country>
    <initial_size>1250</initial_size>
    <panels_remaining>1160</panels_remaining>
  </film>
</media_info>

```

```

<!--Current settings for custom card types.-->
<cards>
  <card>
    <!--General card information for custom card types.-->
    <!--All card thickness values are specified in mils.-->
    <information>
      <type>0</type>
      <zebra_part_number>Custom 1</zebra_part_number>
      <description>Custom 1</description>
      <thickness>30</thickness>
      <mag_stripe>no</mag_stripe>
      <coercivity></coercivity>
      <contact_encode>no</contact_encode>
      <non_contact_encode>no</non_contact_encode>
    </information>

    <!--Temp settings for single-sided and double-sided transfers.-->
    <!--All temperatures and offsets are specified in Celsius.-->
    <transfer_temps>
      <top_single min="0" max="220">185</top_single>
      <top_double min="0" max="220">175</top_double>
      <bot_double min="0" max="220">165</bot_double>
    </transfer_temps>

    <!--Input and output speed settings for single-sided and
    double-sided transfers.-->
    <!--All transfer speeds are specified in inches per second.-->
    <transfer_speeds>
      <input_single min="1.00" max="8.00">1.50</input_single>
      <output_single min="1.00" max="8.00">1.75</output_single>
      <input_double min="1.00" max="8.00">1.50</input_double>
      <output_double min="1.00" max="8.00">1.75</output_double>
    </transfer_speeds>

    <!--Calibration table associated with the card type.-->
    <cal_table min="1" max="2">1</cal_table>
  </card>

  <card>
    <!--General card information for custom card types.-->
    <!--All card thickness values are specified in mils.-->
    <information>
      <type>1</type>
      <zebra_part_number>Custom 2</zebra_part_number>
      <description>Custom 2</description>
      <thickness>30</thickness>
      <mag_stripe>no</mag_stripe>
      <coercivity></coercivity>
      <contact_encode>no</contact_encode>
      <non_contact_encode>no</non_contact_encode>
    </information>

    <!--Temp settings for single-sided and double-sided transfers.-->
    <!--All temperatures and offsets are specified in Celsius.-->
    <transfer_temps>
      <top_single min="0" max="220">185</top_single>
      <top_double min="0" max="220">175</top_double>
      <bot_double min="0" max="220">165</bot_double>
    </transfer_temps>

    <!--Input and output speed settings for single-sided and
    double-sided transfers.-->

```

```

        <!--All transfer speeds are specified in inches per second.-->
        <transfer_speeds>
            <input_single min="1.00" max="8.00">1.50</input_single>
            <output_single min="1.00" max="8.00">1.75</output_single>
            <input_double min="1.00" max="8.00">1.50</input_double>
            <output_double min="1.00" max="8.00">1.75</output_double>
        </transfer_speeds>

        <!--Calibration table associated with the card type.-->
        <cal_table min="1" max="2">1</cal_table>
    </card>
</cards>

<!--Image transfer settings-->
<!--Most transfer settings are not user-settable. Transfer settings are dependent
on the card type specified in the print job with the exception of temperature
offset values. The non-settable values are for factory default only.-->
<transfer_settings>
    <!--Default transfer type: dual-sided or single-sided.-->
    <transfer_type>single</transfer_type>

    <!--Default start-up temperature settings and offsets for single-sided and
double-sided transfers.-->
    <!--All temperatures and offsets are specified in Celsius.-->
    <transfer_temps>
        <top_single min="0" max="220">185</top_single>
        <top_double min="0" max="220">175</top_double>
        <bot_double min="0" max="220">165</bot_double>

        <!--Temperature offsets are user-settable and can be applied to
all transfer temperatures for non-custom cards.-->
        <top_single_offset min="-10" max="10">0</top_single_offset>
        <top_double_offset min="-10" max="10">0</top_double_offset>
        <bot_double_offset min="-10" max="10">0</bot_double_offset>
    </transfer_temps>

    <!--Default start-up input and output speed settings for single-sided and
double-sided transfers. Transfer speeds are specified in inches per second.-->
    <transfer_speeds>
        <input_single min="1.00" max="8.00">1.50</input_single>
        <output_single min="1.00" max="8.00">1.75</output_single>
        <input_double min="1.00" max="8.00">1.50</input_double>
        <output_double min="1.00" max="8.00">1.75</output_double>
    </transfer_speeds>
</transfer_settings>

<!--LCD contrast-->
<lcd_contrast min="1" max="63">32</lcd_contrast>

<!--Cleaning thresholds-->
<cleaning_thresholds>
    <x_direction_card_path min="100" max="5000">4999</x_direction_card_path>
    <y_direction_card_path min="100" max="5000">4999</y_direction_card_path>
    <transfer_rollers min="100" max="20000">19999</transfer_rollers>
</cleaning_thresholds>

<!--The following smart card encoding section specifies only the physical
installation of the internal encoding module.-->
<internal_encoder>
    <comms_channel>usb</comms_channel>
    <contact_encoding>yes</contact_encoding>
    <contactless_encoder>none</contactless_encoder>
</internal_encoder>
</configuration>

```

## Logs

### CleaningLog.xml: Cleaning History

This is the Cleaning History Record of the 32 most recent cleaning events by card count since new or since card count reset if applicable. The log is maintained in a circular buffer; when filled the latest entry replaces the oldest.

<number> = the count of cleaning events 1 is first event and incrementing to the last event  
<cards> = total number of cards which have been printed at the time the cleaning took place

```
<?xml version="1.0" encoding="UTF-8" ?>
- <get_log_cleaning_history>
- <cleaning>
  <number>1</number>
  <cards>5805</cards>
</cleaning>
- <cleaning>
  <number>2</number>
  <cards>5805</cards>
</cleaning>
- <cleaning>
  <number>3</number>
  <cards>5805</cards>
</cleaning>
- <cleaning>
  <number>4</number>
  <cards>5805</cards>
</cleaning>
- <cleaning>
  <number>5</number>
  <cards>6757</cards>
</cleaning>
- <cleaning>
  <number>6</number>
  <cards>6757</cards>
</cleaning>
- <cleaning>
  <number>7</number>
  <cards>6757</cards>
</cleaning>
- <cleaning>
  <number>8</number>
  <cards>6962</cards>
</cleaning>
- <cleaning>
  <number>9</number>
  <cards>7064</cards>
</cleaning>
- <cleaning>
  <number>10</number>
  <cards>7064</cards>
</cleaning>
- <cleaning>
  <number>11</number>
  <cards>7064</cards>
</cleaning>
- <cleaning>
  <number>12</number>
  <cards>7064</cards>
</cleaning>
</get_log_cleaning_history>
```

## ErrorLog.xml: Error History

This is the Error History Record of the 32 most recent error events tabulated in XML form. The log is maintained in a circular buffer; when filled the latest entry replaces the oldest. See [Appendix](#) for the list of error codes.

<number> = the count of errors 1 is the first error and incrementing to the last error  
 <code> = the actual error code which caused the error to be logged  
 <cards> = total number of cards which have been printed at the time the error occurred  
 <print\_lines> = total number of lines printed by the printer at the time the error occurred

```
<?xml version="1.0" encoding="UTF-8" ?>
- <get_log_error_history>
- <error>
  <number>1</number>
  <code>3009</code>
  <cards>7875</cards>
  <print_lines>11290104</print_lines>
</error>
- <error>
  <number>2</number>
  <code>3009</code>
  <cards>7875</cards>
  <print_lines>11290104</print_lines>
</error>
- <error>
  <number>3</number>
  <code>3009</code>
  <cards>7875</cards>
  <print_lines>11290104</print_lines>
</error>
- <error>
  <number>4</number>
  <code>3009</code>
  <cards>7875</cards>
  <print_lines>11290104</print_lines>
</error>
- <error>
  <number>5</number>
  <code>3009</code>
  <cards>7875</cards>
  <print_lines>11290104</print_lines>
</error>
- <error>
  <number>6</number>
  <code>3009</code>
  <cards>7875</cards>
  <print_lines>11290104</print_lines>
</error>
- <error>
  <number>7</number>
  <code>3009</code>
  <cards>7875</cards>
  <print_lines>11290104</print_lines>
</error>
- <error>
  <number>8</number>
  <code>3009</code>
  <cards>7875</cards>
  <print_lines>11290104</print_lines>
</error>
</get_log_error_history>
```

## EventLog.xml: Event History

This is the Event History Record of the most recent 2000 changes in the printer's status including all error conditions no matter what their severity. The log is maintained in a circular 65536 byte buffer; when filled the latest entry replaces the oldest.

<number> = the count of events 1 is the first event and incrementing to the last event  
 <cards> = total number of cards which have been printed at the time the event occurred  
 <description> = description of the event

```
<?xml version="1.0" encoding="UTF-8" ?>
- <get_log_event_history>
- <event>
  <number>1</number>
  <cards>7593</cards>
  <description>Out of cards</description>
</event>
- <event>
  <number>2</number>
  <cards>7593</cards>
  <description>Replenished cards</description>
</event>
- <event>
  <number>3</number>
  <cards>7593</cards>
  <description>Out of cards</description>
</event>
- <event>
  <number>4</number>
  <cards>7593</cards>
  <description>Replenished cards</description>
</event>
- <event>
  <number>5</number>
  <cards>7593</cards>
  <description>Out of cards</description>
</event>
- <event>
  <number>6</number>
  <cards>7593</cards>
  <description>Replenished cards</description>
</event>
- <event>
  <number>7</number>
  <cards>7593</cards>
  <description>Out of cards</description>
</event>
- <event>
  <number>8</number>
  <cards>7593</cards>
  <description>Replenished cards</description>
</event>
- <event>
  <number>9</number>
  <cards>7593</cards>
  <description>Out of cards</description>
</event>
- <event>
  <number>10</number>
  <cards>7593</cards>
  <description>Replenished cards</description>
</event>
</get_log_event_history>
```



# Appendix D

## Sample Code

### List

Inhibit Panel Printing .....	247
ZMotif Printer SDK Sample Code .....	247
ZMotifGraphics SDK Sample Code .....	248
Card Movement .....	249
Move Card to Hold Position .....	249
Move Card to Pre-Defined Position .....	250
Track Current Job to Completion .....	251
Set Printer Error Recovery Mode .....	252
Set Printer Error Recovery Mode to NONE .....	252
Set Printer Error Recovery Mode to MEDIUM .....	253
Set Printer Error Recovery Mode to HIGH .....	254
Apply a Color Profile .....	255

## Inhibit Panel Printing

### ZMotif Printer SDK Sample Code

This example shows how to add the Inhibit Image built by the Graphics SDK (see [Section D.2.2](#)) to the Print Job.

```
private byte[] BuildFrontInhibitImage(byte[] InhibitFrontImage
                                     float ImageXOffset float ImageYOffset
                                     int ImageWidth int ImageHeight
                                     out String errMsg)
{
    errMsg = "";
    try
    {
        byte[] bmpFrontImage = null;

        if (InhibitFrontImage != null)
        {
            // use ZMotifGraphics SDK to build the inhibit image:

            bmpFrontImage = BuildInhibitImage(InhibitFrontImage
                                              ZMotifGraphics.ImageOrientationEnum.Landscape
                                              XOffset YOffset ImageWidth ImageHeight
                                              out errMsg);

            if (bmpFrontImage != null)
            {
                // add inhibit image to print job:

                __job.BuildGraphicsLayers(SideEnum.Front
                                          PrintTypeEnum.Inhibit
                                          (int)ImageXOffset
                                          (int)ImageYOffset
                                          0 -1 GraphicTypeEnum.BMP
                                          bmpFrontImage);

                return bmpFrontImage;
            }
        }
    }
    catch(Exception ex)
    {
        errMsg = ex.Message;
    }
    return null;
}
```

## ZMotifGraphics SDK Sample Code

This example shows how to build the Inhibit Image in the Graphics Buffer.

```
private byte[] BuildInhibitImage(byte[] InhibitImage
                                ZMotifGraphics.ImageOrientationEnum ImageOrientation
                                float xPos float yPos
                                int imageWidth int imageHeight
                                out String errMsg)
{
    ZMotifGraphics graphics = null;
    try
    {
        errMsg = "";

        int dataLen = 0;
        byte[] TheImage = null;
        graphics = new ZMotifGraphics();

        ZMotifGraphics.RibbonTypeEnum RibbonType =
            ZMotifGraphics.RibbonTypeEnum.Inhibit;

        graphics.InitGraphics(0 0 ImageOrientation RibbonType);

        graphics.DrawImage(ref InhibitImage xPos yPos
                           imageWidth imageHeight 0);

        if (InhibitImage != null)
            TheImage = graphics.CreateBitmap(out dataLen);

        return TheImage;
    }
    catch (Exception ex)
    {
        errMsg = ex.Message;
    }
    finally
    {
        graphics.ClearGraphics();
        graphics = null;
    }
    return null;
}
```

## Card Movement

### Move Card to Hold Position

This example demonstrates how to move a card from the Card Feeder to the internal hold position and to determine when card has arrived at hold position.

```
private void MoveCardToHoldPosition()
{
    try
    {
        int actionID = 0;
        string errMsg = "";
        short alarm = 0;

        FeederSourceEnum JobSourceLocation =
            FeederSourceEnum.CardFeeder;
        DestinationTypeEnum JobDestinationLocation =
            DestinationTypeEnum.Hold;

        if (!OpenConnectionToPrinter())
        {
            LogAppend("Failed to open connection to printer");
            return;
        }

        // move the card to the hold position see Section D.3.2:
        alarm = PositionCard(JobSourceLocation
            JobDestinationLocation
            out actionID out errMsg);

        if( (alarm != 0) && (alarm != OUT_OF_CARDS) )
        {
            MessageBox.Show("PositionCard returned alarm = " +
                Convert.ToString(alarm));
        }
        else
        {
            // poll printer for job status see Section D.4:
            if (!WaitForJobToComplete(actionID out errMsg))
            {
                MessageBox.Show(errMsg);
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("ActionID = " + actionID + " Exception: " +
            ex.Message);
    }
    finally
    {
        CloseConnectionToPrinter();
    }
}
```

## Move Card to Pre-Defined Position

This example moves the card from its current location to a pre-defined destination.

```
public short PositionCard(FeederSourceEnum currentLocation
                        DestinationTypeEnum destinationLocation
                        out int actionID out String errMsg)
{
    errMsg = "";
    actionID = 0;
    short alarm = 0;
    try
    {
        Job.JobControl.FeederSource = currentLocation;
        Job.JobControl.Destination = destinationLocation;

        alarm = Job.PositionCard(out actionID);

        return alarm;
    }
    catch (Exception ex)
    {
        errMsg = ex.Message;
    }
    return -1;
}
```

## Track Current Job to Completion

This function determines when a job has completed or failed.

```
// The function periodically polls the printer for job status.
// If the printer returns a job status of "done_ok" this indicates
// successful job completion.
// If the job fails an error string will be returned.

private Boolean WaitForJobToComplete(int actionID out string errMsg)
{
    Boolean bSuccess = false;
    errMsg = "";

    try
    {
        const int MAX_LOOPS = 60;

        int copiesCompleted = 0
        copiesRequested = 0
        alarm = 0
        errorCode = 0;

        string contactStatus = ""
        contactlessStatus = ""
        magStatus = ""
        jobStatus = ""
        cardPosition = ""
        uuidJob = "";

        int loops;
        for (loops = 0; loops < MAX_LOOPS; loops++)
        {
            alarm = Job.GetJobStatus(actionID out uuidJob
                                    out jobStatus
                                    out cardPosition
                                    out errorCode
                                    out copiesCompleted
                                    out copiesRequested
                                    out magStatus
                                    out contactStatus
                                    out contactlessStatus);

            if (jobStatus == "done_ok") // job completed successfully
            {
                bSuccess = true;
                break;
            }
            else if (jobStatus.ToLower().Contains("error")) // job failed
            {
                errMsg = jobStatus;
                break;
            }
            Thread.Sleep(500);
        }

        if (loops >= MAX_LOOPS)
            errMsg = "Job timed out";
    }
    catch (Exception ex)
    {
        bSuccess = false;
        errMsg = ex.Message;
    }
    return bSuccess;
}
```

## Set Printer Error Recovery Mode

This section demonstrates how to set the printer's error recovery mode to each of the three possible levels: none medium and high.

### Set Printer Error Recovery Mode to NONE

This function sets the printer error recovery mode to none. In this mode the printer will *not* attempt to retry a print job following resolution of an error.

None should be chosen if error recovery is to be performed at the application level.

```
private bool SetErrorLevelToNone(string deviceName out string errMsg)
{
    bool result = false;
    errMsg = "";

    try
    {
        job.Open(deviceName);
        job.Device.ErrorControlLevel = ErrorControlLevelEnum.EC_None;
        result = true;
    }
    catch (Exception ex)
    {
        result = false;
        errMsg = ex.Message;
    }
    finally
    {
        job.Close();
    }
    return result;
}
```

## Set Printer Error Recovery Mode to MEDIUM

This function sets the printer error recovery mode to medium. In this mode the printer will automatically retry a print job following resolution of an error.

Medium should be chosen if multiple jobs are to be sent to the printer at the same time.

```
private bool SetErrorLevelToMedium(string deviceName out string errMsg)
{
    bool result = false;
    errMsg = "";

    try
    {
        job.Open(deviceName);
        job.Device.ErrorControlLevel = ErrorControlLevelEnum.EC_Medium;
        result = true;
    }
    catch (Exception ex)
    {
        result = false;
        errMsg = ex.Message;
    }
    finally
    {
        job.Close();
    }
    return result;
}
```

## Set Printer Error Recovery Mode to HIGH

This function sets the printer error recovery mode to high. In this mode the printer will automatically retry a print job following resolution of an error.

High should be chosen if jobs are to be sent to the printer one-at-a-time. In high mode the printer must successfully complete the current job before attempting another job.

```
private bool SetErrorLevelToHigh(string deviceName out string errMsg)
{
    bool result = false;
    errMsg = "";

    try
    {
        job.Open(deviceName);
        job.Device.ErrorControlLevel = ErrorControlLevelEnum.EC_High;
        result = true;
    }
    catch (Exception ex)
    {
        result = false;
        errMsg = ex.Message;
    }
    finally
    {
        job.Close();
    }
    return result;
}
```

## Apply a Color Profile

The following code demonstrates how to build an image with a color profile.

```
private byte[] BuildImageWithColorProfile(byte[] image
    ZMotifGraphics.ImageOrientationEnum ImageOrientation
    ZMotifGraphics.RibbonTypeEnum RibbonType
    String colorProfile out String errMsg)
{
    try
    {
        errMsg = "";

        int dataLen = 0;
        byte[] TheImage = null;

        _graphics.InitGraphics(0 0 ImageOrientation RibbonType);

        if (RibbonType == ZMotifGraphics.RibbonTypeEnum.Color)
            _graphics.ColorProfile = colorProfile;
        else
            _graphics.ColorProfile = string.Empty;

        _graphics.DrawImage(ref image
            ZMotifGraphics.ImagePositionEnum.Centered 1100 700 0);

        if (image != null)
            TheImage = _graphics.CreateBitmap(out dataLen);

        return TheImage;
    }
    catch (Exception ex)
    {
        errMsg = ex.Message;
    }
    finally
    {
        _graphics.ClearGraphics();
        _graphics.ColorProfile = string.Empty;
    }
    return null;
}
```

# Appendix E

# Page Size & Orientation Conventions

## Page and image specifications

There is no single definition of “page” and “image orientation” applicable across the spectrum of printers. Portrait to one printer is landscape to another. Likewise image orientation is arbitrary depending on the application delivering the image to the printer driver. Yet a third factor is the printer driver’s ability to override the application’s definition of wide vs. tall landscape vs. portrait.

The best that can be hoped for in a general purpose ZMotif SDK is a consistent unambiguous definition of two fundamental concepts: the **physical** page and the **logical** page. In ZMotif, image **orientation** is related solely to the logical page.

## Physical page

In ZMotif the physical page is defined differently for roll fed and sheet fed printers. (ID card printers are in the same category as sheet fed printers.) There is no “natural” orientation of the physical page. This depends on the printer itself the currently loaded media choices and the media selected at the printer driver level.

The length of a physical page is defined as its longer dimension the axis along which the longest print line can potentially be applied regardless of image orientation. For a roll-fed printer, this is along the length of the roll; for a sheet-fed printer, it is the longer dimension of the paper, film, or ID card. Width is the dimension 90 degrees to the length.

## Logical page

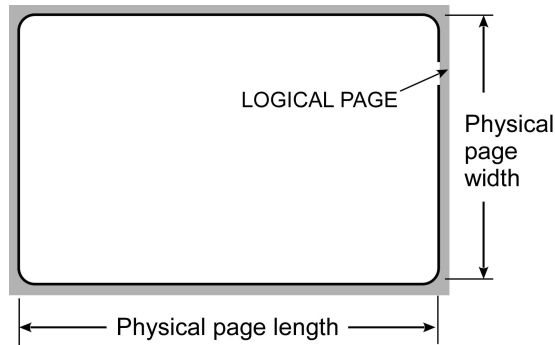
The logical page is the basic “page” concept supported by ZMotif. It is the area that the host assumes to be available for printing based on the media specified.

In ZMotif the orientations and sizes of logical page and physical page are not automatically correlated so some form of image manipulation may be necessary to achieve the desired relationship between the two.

The location of a mag encoding head in an ID card printer for instance usually dictates the orientation of cards in the input hopper; if the head faces up the cards must be loaded with mag stripes down. This may in turn require the image to be rotated 180° so that it prints appropriately relative to the stripe. Be aware that rotation in this sense means rotation of the entire bitmap (and therefore the logical page) in the printer driver see “Printer conventions” below. It has nothing to do with positioning – and possible rotation – of individual components in the application responsible for composing the bitmap.

It is up to the system developer to decide which image manipulation functions will be handled by the host and which will be assigned to the printer. Examples of host functions are: Rotation of the image to fit the physical page reported by the printer; Nesting multiple small logical pages in a single physical page; Tiling a single large logical page over two or more physical pages.

One function that is routinely left to the printer is cropping an over-size logical page to fit the printable area of the physical page. An exception to this is possible in the special case of full-bleed ("edge-to-edge") printers that can have a logical page equal to or larger than the physical page. In the full-bleed example below the physical page is a CR-80 size ID card.



Full bleed logical page vs. physical page

### Logical page size and orientation

Dimensions of the logical page are specified in X and Y pixels. The origin of the X and Y axes is at the upper left corner of the logical page as shown in the following illustrations. The **extents** of X and Y values define the orientation of the logical page and thus the image content – landscape if the X extent is greater than Y portrait if less than Y.

Orientation of the logical page landscape or portrait is the **same as the viewing orientation** intended for the printed product – landscape for a team photo portrait for a chimney. This does **not** imply that the image content of (say) a landscape logical page has to be wider than it is tall although that will often be the case.

One instance where the "wider/taller" convention may not apply is the photo image for a driver license. This could be sent to the printer as a separate entity in which case it would be both smaller than the logical page **and** differently oriented. (Alternatively – and most likely – the photo image could simply be pre-rotated in the card-composing application then placed in a full-page landscape bitmap containing the entire front side image of the license.)

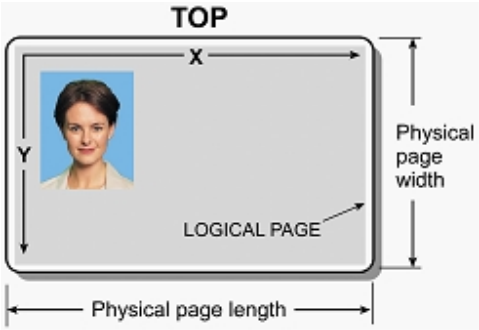


Photo ID card Landscape logical page

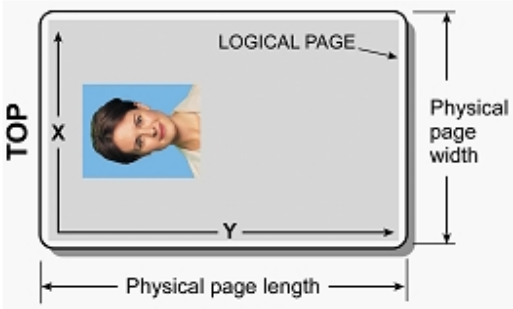
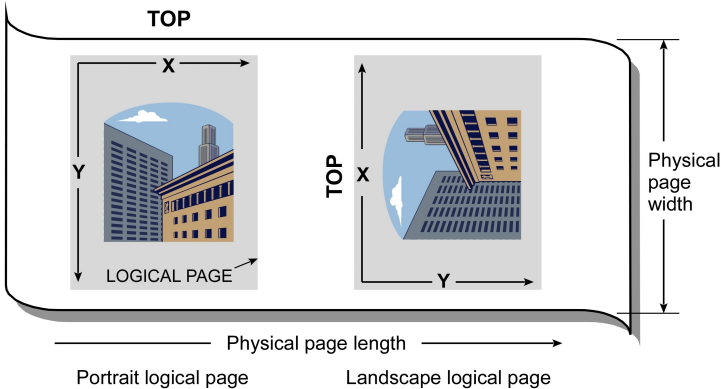
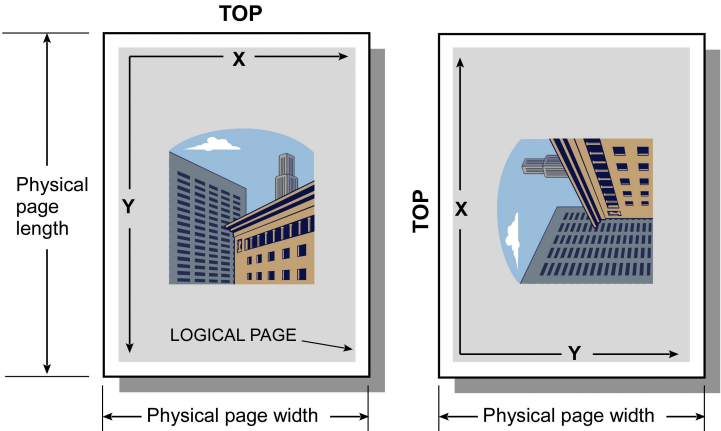


Photo ID card Portrait logical page



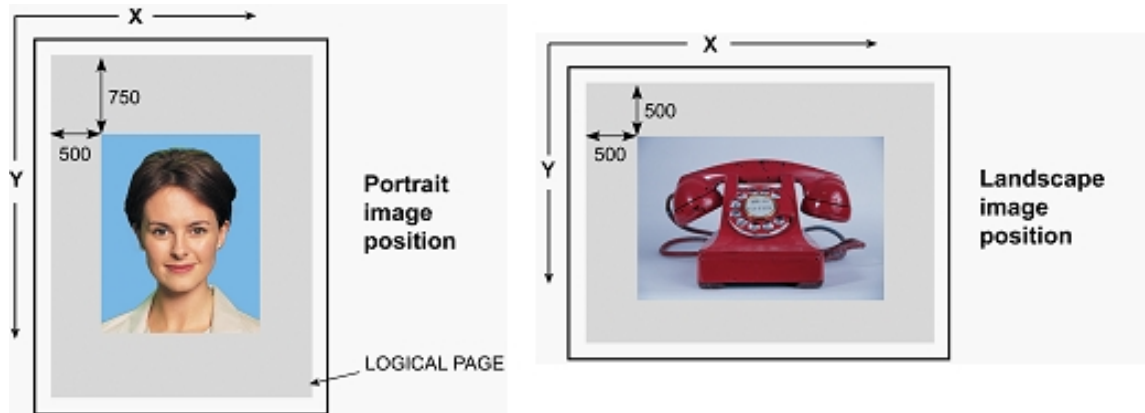
Roll fed media



Sheet fed media

## Image position

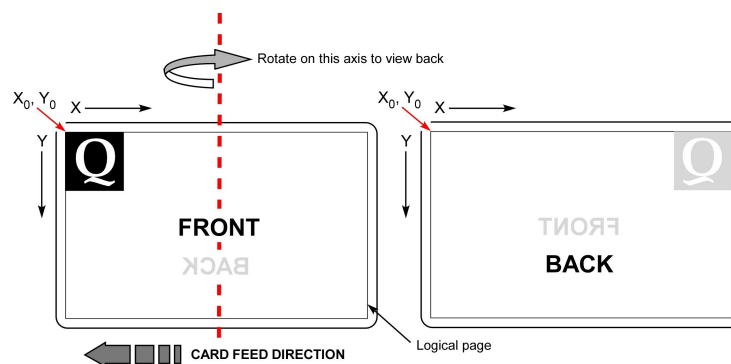
Image position is specified as the offset of the upper left corner of the image after rotation (if any) relative to the upper left corner of the logical page. Offset is measured in X and Y pixels.



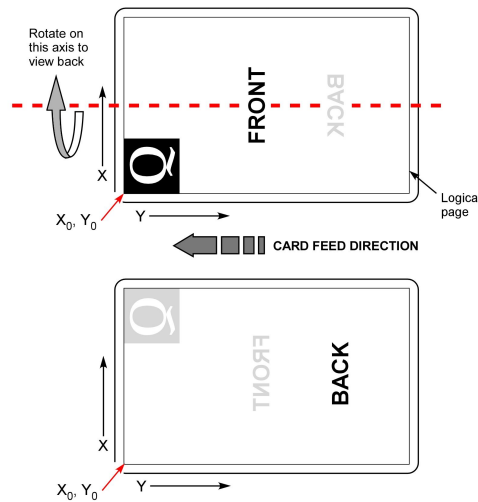
## Printer conventions

Card motion through Zmotif compliant printers is typically right to left viewed from the front of the printer. The logical page origin ( $X_0$   $Y_0$ ) is at the top left corner for a landscape card and bottom left for a portrait. This applies whether the card exits the printer face up or face down. The printers are configured so that the as-viewed page origin of front and back sides is in exactly the same location provided the convention shown below is observed.

**Portrait card viewing convention** – Rotate the card about its long axis to view the back side with the origin at bottom left.



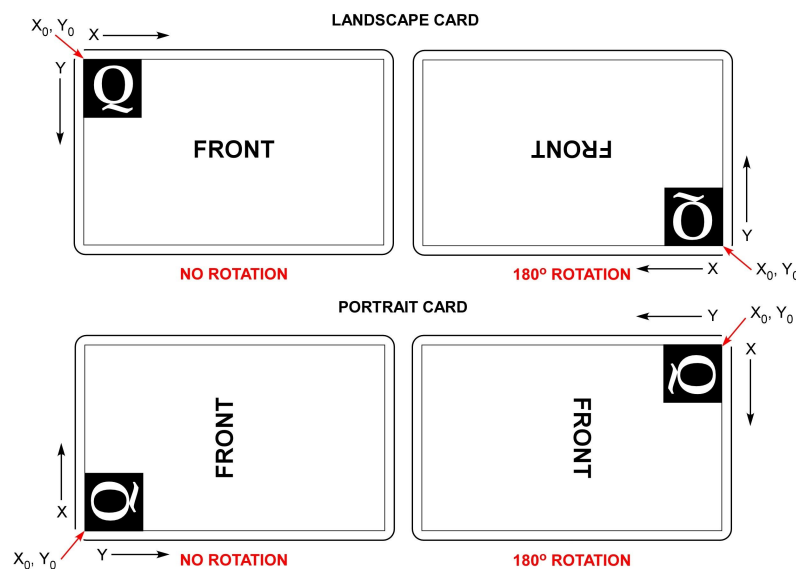
**Landscape card viewing convention** – Rotate the card about its short axis to view the back side with the origin at top left.



## Image rotation

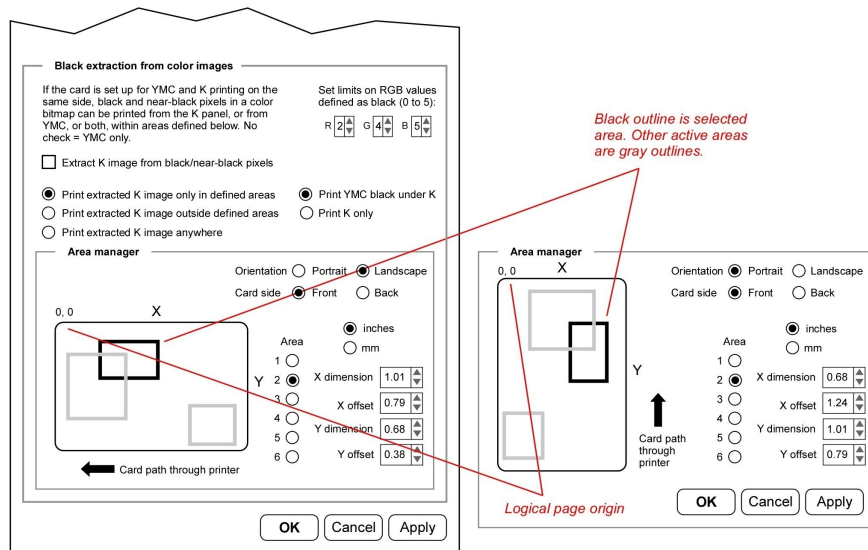
Zmotif compliant printers allow the printed image to be rotated 180 degrees relative to the physical card. Rotation can be applied to both landscape and portrait formats and may be independently applied to front and back sides of the card.

As illustrated below rotation has the effect of shifting the origin of the logical page to the diametrically opposite corner. This is a source of confusion that may be avoided by always visualizing the image composition to be referenced to the "normal" un-rotated page origin. Think of rotation as an override affecting all elements of the composition including those rendered in the printer (which may themselves have been individually rotated to achieve some local effect)..



**Rotation affects the entire image** – It is an override used to position the printed image relative to a magnetic stripe or other physical feature or to allow more convenient viewing.

The top-left 0 0 convention also applies to the "black extraction" capability available in some printer drivers. Black-extraction exclusion zones are referenced to the un-rotated page origin see below



**Black extraction** – If the printer driver allows for black extraction exclusion zones as here these will always be referenced to the un-rotated logical page.

