

ZXP Series 1

ZXP Series 3



ZEBRA

Software Developer Reference Manual

Does Not Require Printer Driver

Copyright

© 2017 ZIH Corp. and/or its affiliates. All rights reserved. ZEBRA and the stylized Zebra head are trademarks of ZIH Corp., registered in many jurisdictions worldwide. All other trademarks are the property of their respective owners.

COPYRIGHTS & TRADEMARKS: For complete copyright and trademark information, go to:
www.zebra.com/copyright

WARRANTY: For complete warranty information, go to: www.zebra.com/warranty

END USER LICENSE AGREEMENT: For complete EULA information, go to: 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

Introduction	1
About This Manual	1
Required Skills	1
Zebra Card Printers	1
Communication Ports	1
SDK Elements	2
Printer	2
SmartCard	2
Installation	2
Card Handling	3
Job	5
Properties	5
Method List	5
Methods	6
BuildGraphicsLayers	6
ClearGraphicsLayers	7
Close	8
EjectCard	9
FlipCard	10
GetBroadcastConfiguration	11
GetDriverName	12
GetJobCountInfo	13
GetJobList	14
GetJobStatus	15
GetPrinters	16
GetSDKVersion	17
GetSDKProductVersion	18
JobCancel	19
JobReprint	20
JobResume	21
JobRetry	22
MagDataOnly	23
Open	24
PositionCard	25
PrintGraphicsLayers	26
PrintGraphicsLayersWithMagData	27
ReadMagData	28
Reset	29
SetBroadcastConfiguration	30
SmartCardDataOnly	31

Job Control	33
Properties	33
Method List	33
Methods	34
GetBlackIntensity	34
GetCyanIntensity	35
GetHalfPanelOffset	36
GetMagentaIntensity	37
GetMonoConvType	38
GetOverlayIntensity	39
GetYellowIntensity	40
SetBlackIntensity	41
SetCyanIntensity	42
SetHalfPanelOffset	43
SetMagentaIntensity	44
SetMonoConvType	45
SetOverlayIntensity	46
SetYellowIntensity	47
SmartCardConfiguration	48
Device	49
Properties	49
Method List	49
Methods	50
BuildOCPMessage	50
ClearOCPMessage	51
DisplayOCPMessage	52
GetConfiguration	53
GetDeviceInfo	54
GetDisplayedOCPMessage	55
GetErrorCount	56
GetMagneticEncoderConfiguration	57
GetNetworkParams	58
GetPrinterStatus	59
GetRibbonParams	60
GetSensorStates	61
GetSensorValues	62
GetSmartCardConfiguration	63
GetStatusMessageString	64
GetTotalCardCount	65
GetXOffset	66
GetYOffset	67
MoveCard	68
SendCommand	69
SetNetworkParams	70
SetXOffset	71
SetYOffset	72
UpgradeEthernetFirmware	73
UpgradeFirmware	74

Deprecated Commands	75
Methods	75
ByteArrayToVariantArray	76
BytePtrToVariantArray	77
IntArrayToVariantArray	78
LongArrayToVariantArray	79
VariantArrayToByteArray	80
VariantArrayToIntArray	81
VariantArrayToLongArray	82
Error Codes	83
Introduction	83
Errors and Alarms	83
Errors	83
Alarms	83
Error Codes and Descriptions	84
ZXP SDK Enumerations	93
Job Enums	93
For ZXP Series 3C only	94



Introduction

About This Manual

This manual contains information for software developers intending to write applications for Zebra ZXP Series 1 and ZXP Series 3 Card Printers. The application programming interface (API) is similar to the ZMotif SDK and provides functions to access card printer features, build and send jobs to the printer, and to track jobs to completion.

This SDK is compatible with the following Windows Operating Systems:

- Windows XP
- Windows Server 2003 and Server 2008
- Windows Vista
- Windows 7 (32/64 bits)
- Windows 8 and 8.1
- Windows 10

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

Required Skills

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

Zebra Card Printers

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

Communication Ports

- USB 2.0
- Ethernet

SDK Elements

Printer

- ZXPPrinter.dll
 - 32 bit dynamic link library
 - calling convention is __stdcall
- The dll is a COM object and requires registration before it can be added as a reference to a software project or deployed for use with a finished software application.
- C# sample code

SmartCard

- For USB, Smart Card encoding is done via the PC/SC API available as part of the Windows operating system.
- For Ethernet, in addition to the PC/SC API, you will need to install the Smart Card Encoding Over Ethernet SDK.

Installation

Directory Structure

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

doc directory contains SDK documentation

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

sample directory contains example applications

System Directories

The SDK can be placed in the system32 folder or can be installed locally with the application.

Example -- XP

(Disk Drive):\WINDOWS\system32\

Card Handling

For Smart Card + Print Job:

1. Open the connection to the printer.
2. Build the smart card encoding + the printing job.
3. Sent the job to the printer.
4. Track the job until the printer indicates the card is at the smart card encoding position.
5. Perform the smart card encoding (requires knowledge of PC/SC API).
6. If encoding is successful, send the JobResume command to the printer; else send the JobCancel command to the printer.
7. Track the job to completion.
8. Close the connection to the printer.



Job

Properties

Boolean

IsOpen (read only) indicates device connection status: returns true if open, false if closed

Float

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

MagDataValidation gets/sets a flag indicating whether or not to validate the user entered magnetic encoding track data (default = true).

Interface

Device returns the Device Interface

JobControl returns the JobControl Interface

Utilities returns the Utilities Interface [Deprecated: Do not use]

Method List

BuildGraphicsLayers.....	6
ClearGraphicsLayers	7
Close.....	8
EjectCard	9
FlipCard	10
GetBroadcastConfiguration.....	11
GetDriverName.....	12
GetJobCountInfo.....	13
GetJobList.....	14
GetJobStatus	15
GetPrinters	16
GetSDKVersion.....	17
GetSDKProductVersion	18
JobCancel.....	19
JobReprint.....	20
JobResume.....	21
JobRetry	22
MagDataOnly	23
Open.....	24
PositionCard	25
PrintGraphicsLayers	26
PrintGraphicsLayersWithMagData	27
ReadMagData.....	28
Reset.....	29
SetBroadcastConfiguration.....	30
SmartCardDataOnly	31
SetHalfPanelOffset	33

Methods

BuildGraphicsLayers

Description: Builds a graphic image into a graphics layer.

Syntax:

```
void BuildGraphicsLayers (
    SideEnum      side,
    PrintTypeEnum  printType,
    GraphicTypeEnum  graphicType,
    object         graphicData )
```

Parameters:

side	[in]specifies the graphic layer's card side (see Appendix for enumeration values)
printType	[in]type of print to perform(see Appendix for enumeration values)
graphicType	[in]image format (see Appendix for enumeration values)
graphicData	[in]image to be printed

Returns: nothing

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

Sample:

```
byte[] graphicData = bitmap to be printed for the layer

job.BuildGraphicsLayers(SideEnum.Front, PrintTypeEnum.Color,
    GraphicTypeEnum.BMP, graphicData);
```

ClearGraphicsLayers

Description: Erases all data from the graphic layers.

Syntax: `void ClearGraphicsLayers()`

Parameters: none

Returns: nothing

Sample: `job.ClearGraphicsLayers();`

Close

Description: Closes the connection to a ZXP-1 or ZXP-3 Printer.

Syntax: void Close()

Parameters: none

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.

EjectCard

Description: Instructs printer to eject a card.

Syntax: `short EjectCard()`

Parameters: none

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

Sample: `short alarm = Job.EjectCard();`

FlipCard

Description: Instructs the printer to flip a card (ZXP-3 dual-sided configuration only).

Syntax: short FlipCard()

Parameters: none

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

Sample: short alarm = Job.FlipCard();

GetBroadcastConfiguration

Description: Returns a printer's 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]maximum number of devices allowed

Returns: nothing

Sample:

```
int retries = 0;
float timeout = 0.0;
int maxDevices = 0;

job.GetBroadcastConfiguration(out retries, out timeout, out maxDevices);
```

GetDriverName

NOTE: Printer driver must be installed prior to using this method.

Description: Retrieves the printer name from the printer driver.

Syntax:

```
void GetDriverName(  
    string          deviceName,  
    out string      driverName )
```

Parameters:

deviceName	[in]serial number or IP address of the ZXP Printer
driverName	[out] name of ZXP Printer assigned by printer driver

Returns: Nothing

Sample:

```
string deviceName = "Serial Number of ZXP Printer";  
string driverName = string.Empty;  
  
job.GetDriverName(deviceName, out driverName);
```

GetJobCountInfo

Description: Returns selected information regarding a printer's jobs' statuses.

Syntax:

```
short GetJobCountInfo(  
    out int jobsPending,  
    out int jobsActive,  
    out int jobsComplete,  
    out int jobErrors,  
    out int jobsTotal)
```

Parameters:

jobsPending	[out]number of pending jobs
jobsActive	[out]number of active jobs
jobsComplete	[out]number of completed job
jobErrors	[out]number of jobs with errors
jobsTotal	[out]total number of jobs

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

Sample:

```
int jobsPending = 0;  
int jobsActive = 0;  
int jobsComplete = 0;  
int jobErrors = 0;  
int jobsTotal = 0;  
  
short alarm = Job.GetJobCountInfo(out jobsPending, out jobsActive,  
    out jobsComplete, out jobErrors,  
    out jobsTotal);
```

GetJobList

Description: Returns a list of the printer's jobs.

Syntax: short GetJobList(out object jobList)

Parameters: jobList [out]string array containing current list of jobs; each string item will be formatted as: "actionID, uuid, status"

Example: "ActionID: 13, UUID: 86b8d6bc-66f8-4758-acd9-7a3036901094, Status: done_ok"

"ActionID: 14, UUID: 593839fc-f889-44d0-8df4-1ee0880695f9, Status: in_progress"

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

Sample:

```
try
{
    short alarm = job.GetJobList(out objJobList);

    if (objJobList != null)
    {
        Array array = (Array)objJobList;
        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: Returns a job's status.

Syntax:

```
short GetJobStatus(
    int          actionID,
    out string   printingStatus,
    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
printingStatus	[out]present job status
errorCode	[out]error code
copiesCompleted	[out]number of copies complete
copiesRequested	[out]number of copies requested
magStatus	[out]magnetic encoding status
contactStatus	[out]contact status
contactlessStatus	[out]contactless 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",
- **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](#))

Sample:

```
while (true)
{
    short alarm = job.GetJobStatus(actionID, out printingStatus,
                                   out errorCode, out copiesCompleted,
                                   out copiesRequested, out magStatus,
                                   out contactStatus, out contactlessStatus);
    ( ... check appropriate status condition(s) ...)
    if required condition met
        break;
```

GetPrinters

Description: Returns 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 for enumeration values)
printerList	[out]printer list as 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:

```
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;
```

GetSDKVersion

Description: Returns 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:

```
byte major = 0;  
byte minor = 0;  
byte build = 0;  
byte revision = 0;  
  
job.GetSDKVersion( out major, out minor, out build, out revision );
```

GetSDKProductVersion

Description: Returns 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 );
```


JobCancel

Description: Cancels a job.

Syntax: `short JobCancel(int actionID)`

Parameters: `actionID` [in]a job's identifier provided by the printer when the job is sent to the printer.

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

Note: `actionID = 0` cancels all jobs

Sample: `short alarm = job.JobCancel (actionID) ;`

JobReprint

Description: Reprints the last job.

Syntax: `short JobReprint(int copies)`

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

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

Sample:

```
int copies = 1;  
short alarm = job.JobReprint ( copies );
```

JobResume

Description: Resumes a suspended job.

Syntax: `short JobResume()`

Parameters: none

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

Sample: `short alarm = job.JobResume();`

JobRetry

Description: Retries the last job performed by the printer.

Syntax: `short JobRetry()`

Parameters: none

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

Sample: `short alarm = job.JobRetry();`

MagDataOnly

Description: Sends a magnetic encoding job to the printer. Only magnetic coding of the card's track(s) is performed; no printing takes place.

Syntax:

```
short MagDataOnly(
    int copies,
    string      track1,
    string      track2,
    string      track3,
    out int     actionID)
```

Parameters:

copies	[in]number of cards to encode
track1	[in]magnetic data for track 1 null or "" indicates no data to encode
track2	[in]magnetic data for track 2 null or "" indicates no data to encode
track3	[in]magnetic data for track 3 null or "" indicates no data to encode
actionID	[out]returned by a ZXP-1 or ZXP-3 printer identifying a job

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

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 **Destination** and **Feeder Source** in [Properties](#) on page 33.

```
try
{
    int copies = 1;
    int actionID = 0;
    string track1 = "TRACK1DATA";
    string track2 = "22222222";
    string track3 = "33333333";

    short alarm = job.MagDataOnly (copies, track1, track2, track3,
                                   out actionID);
}
catch (Exception ex)
{
    errMsg = ex.Message;
```

Open

Description: Establishes a connection to a ZXP-1 or ZXP-3 Printer.

Syntax: short Open(string deviceName)

Parameters: deviceName [in]Serial Number of the printer, printer driver name for the printer, or IP Address of the printer

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

Sample:

//USB Connection:

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

//Ethernet Connection:

```
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:

```
try
{
    //printer driver name
    string deviceName = "Zebra ZXP Series 3 USB Card Printer";
    short alarm = job.Open(deviceName);
}
catch (Exception ex)
{
    string errMsg = ex.Message;
}
```

//Printer driver installed/Ethernet connection:

```
try
{
    //printer driver name
    string deviceName = "Zebra ZXP Series 3 Network Card Printer";
    short alarm = job.Open(deviceName);
}
catch (Exception ex)
{
    string errMsg = ex.Message;
}
```

PositionCard

Description: Moves card from a specified source location to a specified destination.

Syntax: short PositionCard(out int actionID)

Parameters: actionID [out] returned by a ZXP-1 or ZXP-3 printer
identifying a job

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

Sample 1:

```
int actionID = 0;
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);
```

PrintGraphicsLayers

Description: Prints all layers created by BuildGraphicsLayers.

Syntax: short PrintGraphicsLayers(
 int copies,
 out int actionID)

Parameters: copies [in]number of copies to print
 actionID [out]returned by a ZXP-1 or ZXP-3 printer
 identifying a job

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

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 **Destination and Feeder Source in [Properties](#)** on page 33.

```
try
{
    int copies    = 1;
    int actionID = 0;

    short alarm = job.PrintGraphicsLayers(copies, out actionID);
}
catch (Exception ex)
{
    errMsg = ex.Message;
```


PrintGraphicsLayersWithMagData

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

Syntax:

```
short PrintGraphicsLayersWithMagData(
    int             copies,
    string           track1,
    string           track2,
    string           track3,
    out int          actionID)
```

Parameters:

copies	[in]number of cards to print and encode
track1	[in]magnetic data for track 1 null or "" indicates no data to encode
track2	[in]magnetic data for track 2 null or "" indicates no data to encode
track3	[in]magnetic data for track 3 null or "" indicates no data to encode
actionID	[out]returned by a ZXP-1 or ZXP-3 printer identifying a job

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

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 **Destination and Feeder Source in [Properties](#)** on page 33.

```
try
{
    int copies           = 1;
    int actionID         = 0;
    string track1Data    = "ABCDEFGH";
    string track2Data    = "12345678";
    string track3Data    = "87654321";

    short alarm = job.PrintGraphicsLayersWithMagData(copies, track1Data,
                                                    track2Data, track3Data, out actionID);
}
catch (Exception ex)
{
    string errMsg = ex.Message;
```

ReadMagData

Description: Instructs the printer to read and return the data from one or more magnetic tracks of a card.

Syntax:

```
short ReadMagData(
    DataSourceEnum    tracksToRead,
    out string        track1,
    out string        track2,
    out string        track3,
    out int           actionID);
```

Parameters:

tracksToRead	[in] defines the track(s) to be read (see Appendix for enumeration values)
track1	[out] Track 1 data
track2	[out] Track 2 data
track3	[out] Track 3 data
actionID	[out] returned by a ZXP-1 or ZXP-3 printer identifying a job

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

Sample:

```
try
{
    //read all three magnetic tracks:
    DataSourceEnum tracks = DataSourceEnum.Track1Data |
        DataSourceEnum.Track2Data | DataSourceEnum.Track3Data;
    string track1 = string.Empty;
    string track2 = string.Empty;
    string track3 = string.Empty;
    int actionID = 0;
    //Assign all three tracks to be read
    Job.JobControl.DataSource = tracks;
    short alarm = job.ReadMagData(tracks, out track1, out track2, out track3,
        out actionID);
}
catch (Exception ex)
{
    errMsg = ex.Message;
}

EIN example demonstrating one possible way to work with EIN data:
try
{
    string Track1Data =
    string.Empty;
    string Track3Data = string.Empty;
    string EIN = string.Empty;
    int actionID = 0;

    //assigns the track number to be read: 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);
}
catch(Exception ex)
{
    string errMsg = ex.Message;
}

//insert your logic here for processing the EIN data returned from the card
```

Reset

Description: Orders a printer to perform a reboot.

Syntax: `short Reset()`

Parameters: `none`

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

Sample: `short alarm = job.Reset();`

SetBroadcastConfiguration

Description: Assigns a printer's 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]maximum number of devices allowed

Returns: nothing

Sample:

```
int retries = 10;  
float timeout = 30.0;  
int maxDevices = 20;  
  
job.SetBroadcastConfiguration(retries, timeout, maxDevices);
```

SmartCardDataOnly

Description: Sends a smart card encode job to the printer (ZXP-3 only). The printer will position the card for smartcard encoding; however, no smartcard encoding takes place. This must be done outside of the printer SDK using the PC/SC Programming API to interface directly with the smartcard module within the printer.

Syntax:

```
short SmartCardDataOnly(
    int          copies,
    out int      actionID)
```

Parameters:

copies	[in] number of cards to encode
actionID	[out] returned by a ZXP-3 printer identifying a job

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

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

Sample:

```
try
{
    int copies = 1;
    int actionID = 0;
    short alarm = job.SmartCardDataOnly(copies, out actionID);

    // PC/SC smart card code goes here

    // pseudocode example
    if smart card encoding is successful
    {
        job.JobResume();
    }
    else
        job.JobCancel();
}
catch (Exception ex)
{
    errMsg = ex.Message;
```



Job Control

Properties

Boolean

DeleteAfter	gets/sets a flag indicating whether or not a job is to be deleted from the printer's current job list upon completion.
MagVerification	gets/sets a flag indicating whether or not verification of data is performed following magnetic encoding of a card.

Enumeration

DataSource	gets / sets DataSourceEnum indicating from which magnetic track(s) data is to be read (see enumeration).
Destination	gets / sets DestinationTypeEnum defining a card's location at the completion of a job (see enumeration).
FeederSource	gets / sets FeederSourceEnum defining a card's location at the beginning of a job (see enumeration).
MagCoercivity	gets / sets MagCoercivityEnum defining the type of magnetic coercivity for a card (see enumeration).
MagDataValidation	gets/sets a flag indicating whether or not to validate the user entered magnetic encoding track data (default = true).
MagEncodingType	gets / sets MagEncodingTypeEnum to be used for a card to be encoded (see enumeration).

Method List

GetBlackIntensity	34
GetCyanIntensity	35
GetHalfPanelOffset	36
GetMonoConvType	38
GetOverlayIntensity	39
GetYellowIntensity	40
SetBlackIntensity	41
SetCyanIntensity	42
SetHalfPanelOffset	43
SetMonoConvType	45
SetOverlayIntensity	46
SetYellowIntensity	47
SmartCardConfiguration	48

Methods

GetBlackIntensity

Description: Returns the current black intensity value.

Syntax:

```
void GetBlackIntensity (
    SideEnum    side
    out short    value)
```

Parameters:

side	[in]defines the card side for which the value should be returned: Back Front
value	[out]current black intensity value

Returns: nothing

Sample:

```
short value = 0;
job.JobControl.GetBlackIntensity(SideEnum.Back, out value);
```


GetCyanIntensity

Description: Returns the current cyan intensity value.

Syntax:

```
void GetCyanIntensity (
    SideEnum    side
    out short    value)
```

Parameters:

side	[in]defines the card side for which the value should be returned:
	Back
	Front
value	[out]current cyan intensity value

Returns: nothing

Sample:

```
short value = 0;
job.JobControl.GetCyanIntensity(SideEnum.Back, out value);
```

GetHalfPanelOffset

Description: Returns the user set half panel offset value. (default = -1)

Syntax:

```
void GetHalfPanelOffset(  
    SideEnum    side,  
    out short    value)
```

Parameters:

side	[in] defines the card side for which the value should be returned
	Back
	Front
value	[out] current user set half panel offset value

Returns: nothing

Sample:

```
short value = 0;  
job.JobControl.GetHalfPanelOffset(SideEnum.Back, out value);
```

Note: The default value of -1 indicates that the SDK should attempt to automatically determine the appropriate offset values.

GetMagentaIntensity

Description: Returns the current magenta intensity value.

Syntax:

```
void GetMagentaIntensity (  
    SideEnum    side  
    out short    value)
```

Parameters:

side	[in]defines the card side for which the value should be returned: Back Front
value	[out]current magenta intensity value

Returns: nothing

Sample:

```
short value = 0;  
job.JobControl.GetMagentaIntensity(SideEnum.Back, out value);
```

GetMonoConvType

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

Syntax:

```
void GetMonoConvType (
    SideEnum          side
    out MonoConvTypeEnum convType)
```

Parameters:

side	[in]defines the card side for which the value should be returned: Back Front
value	[out]current monochrome conversion type: MonoBarcode MonoDiffusion MonoHalftone MonoText

Returns: nothing

Sample:

```
MonoConvTypeEnum convType;
job.JobControl.GetMonoConvType(SideEnum.Back, out convType);
```

GetOverlayIntensity

Description: Returns the current overlay intensity value:

Syntax: void GetOverlayIntensity(
 SideEnum side
 out short value)

Parameters: side [in] defines the card side for which the value should be returned
 Back
 Front
 value [out] the current overlay intensity value (-100 to 100)

Returns: nothing

Sample: short value = 0;
 job.JobControl.GetOverlayIntensity(SideEnum.Front, out value)

GetYellowIntensity

Description: Returns the current yellow intensity value.

Syntax:

```
void GetYellowIntensity (
    SideEnum    side
    out short    value)
```

Parameters:

side	[in]defines the card side for which the value should be returned: Back Front
value	[out]current yellow intensity value

Returns: nothing

Sample:

```
short value = 0;
job.JobControl.GetYellowIntensity(SideEnum.Back, out value);
```

SetBlackIntensity

Description: Assigns the black intensity level.

Syntax:

```
void SetBlackIntensity (
                        SideEnum    side
                        short        value)
```

Parameters:

side	[in]defines the card side to which the value should be applied: Back Front
value	[in]the new black intensity value (0-10)

Returns: nothing

Sample:

```
short value = 10;
job.JobControl.SetBlackIntensity(SideEnum.Back,  value);
```

SetCyanIntensity

Description: Assigns the cyan intensity level.

Syntax:

```
void SetCyanIntensity (
                        SideEnum    side
                        short        value)
```

Parameters:

side	[in]defines the card side to which the value should be applied: Back Front
value	[in]the new cyan intensity value (0-10)

Returns: nothing

Sample:

```
short value = 10;
job.JobControl.SetCyanIntensity(SideEnum.Back, value);
```


SetHalfPanelOffset

Description: Assigns the half panel offset value. (default = -1)

Syntax:

```
void SetHalfPanelOffset(  
    SideEnum    side,  
    short       value)
```

Parameters:

side	[in] defines the card side to which the value should be applied: Back Front
------	---

value	[in] the half panel offset value
-------	----------------------------------

Returns: nothing

Sample:

```
short value = 547;  
job.JobControl.SetHalfPanelOffset(SideEnum.Back, value);
```

Note: The default value of -1 indicates that the SDK should attempt to automatically determine the appropriate offset values.

SetMagentaIntensity

Description: Assigns the magenta intensity level.

Syntax:

```
void SetMagentaIntensity (
                        SideEnum    side
                        short        value)
```

Parameters:

side	[in]defines the card side to which the value should be applied:
	Back
	Front
Value	[in]the new magenta intensity value (0-10)

Returns: nothing

Sample:

```
short value = 0;
job.JobControl.SetMagentaIntensity(SideEnum.Back, value);
```

SetMonoConvType

Description: Assigns the type of monochrome conversion to use for the designated side of the card.

Syntax:

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

Parameters:

side	[in]defines the card side to which the value should be applied:	
	Back	
	Front	
value	[in]monochrome conversion type to use:	MonoBarcode
	MonoDiffusion	
	MonoHalftone	
	MonoText	

Returns: nothing

Sample:

```
MonoConvTypeEnum convType = MonoConvTypeEnum.MonoText;
job.JobControl.SetMonoConvType(SideEnum.Back, convType);
```

SetOverlayIntensity

Description: Sets the current overlay intensity value:

Syntax: `void SetOverlayIntensity(
 SideEnum side
 short value)`

Parameters: `side` `[in]` defines the card side for which the value should be applied
 `Back`
 `Front`
 `value` `[in]` the new overlay intensity value (-100 to 100)

Returns: `nothing`

Sample: `short value = 10;
 job.JobControl.SetOverlayIntensity(SideEnum.Back, value);`

SetYellowIntensity

Description: Assigns the yellow intensity value.

Syntax:

```
void SetYellowIntensity (
                        SideEnum    side
                        short        value)
```

Parameters:

side	[in]defines the card side to which the value should be applied: Back Front
value	[in]the new yellow intensity value (0-10)

Returns: nothing

Sample:

```
short value = 0;
job.JobControl.SetYellowIntensity(SideEnum.Back,  value);
```

SmartCardConfiguration

Description: Defines the type of smartcard configuration and its location on a card.

Syntax:

```
void SmartCardConfiguration(  
    SideEnum side  
    SmartCardTypeEnum smartCard);
```

Parameters:

side	[in]defines the card location of the smart card configuration: Back Front
smartCard	[in]defines the type of smartcard configuration: Contact Contactless None

Returns: nothing

Sample:

```
job.JobControl.SmartCardConfiguration(SideEnum.Back,  
    SmartCardTypeEnum.Contactless);
```

Device

Properties

Boolean

CardPreFeed gets / sets a flag indicating whether or not the printer's card pre-feed mode is enabled / disabled.

Byte

MagVerificationThreshold gets/sets the magnetic read verification threshold value

Enumerations

ATMMode gets / sets ATMModeEnum defining the printer's current operational mode (see).

PrintCapability gets the TransferTypeEnum defining the printer as single side or dual side print capable (see).

PrintOptimizationMode (ZXP Series 3C only)
gets/sets the print optimization mode (see)

Integer

HeadResistance gets /sets the head resistance value (default value is 0).

SmartCardOffset gets /sets the smart card x offset position value.

Short

EndOfPrint gets / sets the page number of the last page to print (used with printer driver).

MonoBias gets / sets the mono bias level.

Method List

BuildOCPMessage	50
ClearOCPMessage	51
DisplayOCPMessage	52
GetConfiguration	53
GetDeviceInfo	54
GetDisplayedOCPMessage	55
GetErrorCount	56
GetMagneticEncoderConfiguration	57
GetNetworkParams	58
GetPrinterStatus	59
GetRibbonParams	60
GetSensorStates	61
GetSensorValues	62
GetSmartCardConfiguration	63
GetStatusMessageString	64
GetTotalCardCount	65
GetXOffset	66
GetYOffset	67
MoveCard	68
SetNetworkParams	70
SetXOffset	71
SetYOffset	72
UpgradeEthernetFirmware	73
UpgradeFirmware	74

Methods

BuildOCPPMessage

Description: Builds a message to be displayed on the printer's OCP.

Syntax:

```
void BuildOCPPMessage (  
    string message,  
    OCPDisplayModeEnum format)
```

Parameters:

message	[in]message to be displayed
format	[in]display format of message: Blink Normal Scroll

Returns: nothing

Sample:

```
//define a blinking message that instructs the user to insert a smartcard  
  
string message = "Insert Smartcard";  
job.Device.BuildOCPPMessage(message, OCPDisplayModeEnum.Blink);
```


ClearOCPMessage

Description: Clears a message being displayed on the printer's OCP.

Syntax: short ClearOCPMessage()

Parameters: none

Returns: error code (see)

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

DisplayOCPPMessage

Description: Displays the message created by BuildOCPPMessage on the printer's OCP.

Syntax: short DisplayOCPPMessage()

Parameters: none

Returns: error code (see)

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

GetConfiguration

Description: Returns an xml document containing a printer's current configuration.

Syntax: short GetConfiguration (out string xmlConfig)

Parameters: xmlConfig [out]document containing configuration information

Returns: error code (see)

Sample: string xmlConfig = string.Empty;
short alarm = job.Device.GetConfiguration(out xmlConfig);

Example: XML Configuration Document

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <ethernet>
    <dhcp>unknown</dhcp>
    <ip_address>10.1.22.16</ip_address>
    <subnet_mask>255.255.255.0</subnet_mask>
    <gateway>10.1.22.1</gateway>
  </ethernet>
  <imaging_parameters>
    <printhead_resistance>3000</printhead_resistance>
  </imaging_parameters>
  <mech_adjustments>
    <card_x_offset_front>4</card_x_offset_front>
    <card_x_offset_back>4</card_x_offset_back>
    <card_y_offset_front>20</card_y_offset_front>
    <card_y_offset_back>20</card_y_offset_back>
    <card_end_of_print>11</card_end_of_print>
    <card_smart_card_x_offset>387</card_smart_card_x_offset>
  </mech_adjustments>
  <lcd>
    <contrast>10</contrast>
    <brightness>2</brightness>
  </lcd>
  <cleaning_thresholds>
    <x_direction_card_path>5000</x_direction_card_path>
  </cleaning_thresholds>
  <encoder>internal</encoder>
</configuration>
```

GetDeviceInfo

Description: Returns a printer's 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 )
```

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

Returns: error code (see)

Sample:

```
string vendor = string.Empty;
string model = string.Empty;
string serialNumber = string.Empty;
string MAC = string.Empty;
string headSerialNumber = string.Empty;
string oemCode = string.Empty;
string fwVersion = string.Empty;

short alarm = job.Device.GetDeviceInfo(out vendor,
                                       out model,
                                       out serialNumber,
                                       out MAC,
                                       out headSerialNumber,
                                       out oemCode,
                                       out fwVersion);
```

GetDisplayedOCPMessage

Description: Returns the message currently displayed on the printer's OCP.

Syntax: `short GetDisplayedOCPMessage(out string message)`

Parameters: `message` [out]current message displayed on OCP

Returns: error code (see)

Sample: `string message = string.Empty;
short alarm = job.Device.GetDisplayedOCPMessage(out message);`

GetErrorCount

Description: Returns the number of errors encountered by a printer.

Syntax: `short GetErrorCount (out int errorCount)`

Parameters: `errorCount` `[out]`number of errors

Returns: error code (see)

Sample: `int errorCount = 0;`
 `short alarm = job.Device.GetErrorCount(out errorCount);`

GetMagneticEncoderConfiguration

Description: Returns a printer's magnetic encoder configuration.

Syntax:

```
short GetMagneticEncoderConfiguration (
                                out string    headType
                                out string    stripeLocation )
```

Parameters:

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

Returns: error code (see)

Sample:

```
string headType = string.Empty;
string stripeLocation = string.Empty;

short alarm = job.Device.GetMagneticEncoderConfiguration(
                                out headType,
                                out stripeLocation);
```

GetNetworkParams

Description: Returns a printer's current network parameters.

Syntax:

```
short GetNetworkParams (
    out string      MAC
    out string      ipAddress
    out string      subMask
    out string      gateway
    out bool        dhcpEnabled )
```

Parameters:

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

Returns: error code (see)

Sample:

```
string MAC = string.Empty;
string ipAddress = string.Empty;
string subMask = string.Empty;
string gateway = string.Empty;
bool dhcpEnabled = false;

short alarm = job.Device.GetNetworkParams(out MAC,
                                           out ipAddress,
                                           out subMask,
                                           out gateway,
                                           out dhcpEnabled);
```


GetPrinterStatus

Description: Returns a printer's current 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
jobsTotal	[out]number of jobs processed
nextActionID	[out]next job's action ID

Status:

"initializing"	"idle"	"standby"
"printing"	"alarm_handling"	"offline" "canceling" "temp_out_of_range"
"mag_ops" "contact_ops"		"contactless_ops" "config_data" "job_data"
"diagnostic_mode" "insert_card"		

Returns: error code (see)

Sample:

```
string status = string.Empty;
int error = 0;
int jobsPending = 0;
int jobsActive = 0;
int jobsComplete = 0;
int jobsErrors = 0;
int jobsTotal = 0;
int nextActionID = 0;

short alarm = job.Device.GetPrinterStatus(out status, out error, out jobsPending,
    out jobsActive,
    out jobsComplete,
    out jobErrors,
    out jobsTotal,
    out nextActionID);
```

GetRibbonParams

Description: Returns a printer's installed ribbon information.

Syntax:

```
short GetRibbonParams (
    out int         type
    out string      description
    out string      oemCode
    out int         initialSize
    out int         panelsRemaining )
```

Parameters:

type	[out]type of ribbon
description	[out]ribbon type's description
oemCode	[out]OEM part number
initialSize	[out]ribbon type's total number of panels
panelsRemaining	[out]ribbon type's number of unused panels

Returns: error code (see)

Sample:

```
int type           = 0;
string description = string.Empty;
string oemCode     = string.Empty;
int initialSize    = 0;
int panelsRemaining = 0;

short alarm = job.Device.GetRibbonParams(out type,
                                         out description,
                                         out oemCode,
                                         out initialSize,
                                         out panelsRemaining);
```

GetSensorStates

Description: Returns a printer's sensors' states.

Syntax: short GetSensorStates(out object sensorStates)

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

Example of sensors and their states:

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

Returns: error code (see)

Sample:

```
try
{
    object objSensorStates = null;
    short 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: Returns a printer's sensors' values.

Syntax: short GetSensorValues (out object sensorValues)

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

Example of sensors and their values:

"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)

Sample:

```
try
{
    object objSensorValues = null;
    short 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: Returns a printer's smart card configuration.

Syntax: short GetSmartCardConfiguration(out string encoderType)

Parameters: encoderType [out]the type of encoder installed in the
 printer

Returns: error code (see)

Sample: string encoderType = string.Empty;
 short alarm = job.Device.GetSmartCardConfiguration(
 out encoderType);

GetStatusMessageString

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

Syntax: `string GetStatusMessageString (int statusCode)`

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

Returns: error code (see)

Sample: `int statusCode = 4016; //example of status code. 4016 = out of cards. Note: The statusCode value would be returned from an SDK function.`

```
string message = job.Device.GetStatusMessageString(statusCode);
```

GetTotalCardCount

Description: Returns the total card count since last reset.

Syntax: `short GetTotalCardCount (out int cardCount)`

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

Returns: error code (see)

Sample:

```
int cardCount = 0;  
short alarm = job.Device.GetTotalCardCount(out cardCount);
```

GetXOffset

Description: Returns the X offset for the specified side of the card.

Syntax:

```
short GetXOffset (
    SideEnum    side,
    out short    xOffset )
```

Parameters:

side	[in]defines the side of the card; see Appendix C
xOffset	[out]x offset for the designated card side

Returns: error code (see)

Sample:

```
short xOffset = 0;
short alarm = job.Device.GetXOffset(SideEnum.Back, out xOffset);
```


GetYOffset

Description: Returns the Y offset for the specified side of the card.

Syntax:

```
short GetYOffset (
    SideEnum    side,
    out short    yOffset )
```

Parameters:

side	[in]defines the side of the card; see Appendix C
yOffset	[out]y offset for the designated card side

Returns: error code (see)

Sample:

```
short yOffset = 0;
short alarm = job.Device.GetYOffset(SideEnum.Back, out yOffset);
```

MoveCard

Description: Moves card forward or reverse within the printer the specified number of steps.

Syntax: short MoveCard (CardDirectionEnum direction, int steps)

Parameters: direction [in]defines the card movement direction:
 Forward
 Reverse
 steps [in]the number of steps to move

Returns: error code (see)

Sample: int steps = 10;
 short alarm = job.MoveCard(CardDirectionEnum.Forward, steps);

SendCommand

Description: Provides the ability to send limited "EPCL" commands to the printer.

Syntax: short SendCommand(string command, out string commandResponse)

Parameters: command [in]"EPCL" command to be sent to printer commandResponse [out]Printer's response to command received

Returns: error code (see)

Sample Code:

```
string command = "V"; //version command
string commandResponse = string.Empty;

short alarm = job.Device.SendCommand(command, out commandResponse);
```

SetNetworkParams

Description: Assign network parameters to a printer.

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 defines dhcp configuration: BoolTypeEnum.NoChange, BoolTypeEnum.False_BT, BoolTypeEnum.True_BT

Returns: error code (see)

Note: Any of the arguments can be null; if a property is null, that property is not set. Instead, it retains its current value.

Sample:

```
string ipAddress = "127.0.0.1";
string subMask = "255.255.255.0";
string gateway = "127.0.0.1";

short alarm = job.Device.SetNetworkParams(ipAddress, subMask,
    gateway, BoolTypeEnum.True_BT);
```

SetXOffset

Description: Assigns the X offset for the specified side of the card.

Syntax:

```
short SetXOffset (
                SideEnum      side,
                short          xOffset )
```

Parameters:

side	[in]defines the side of the card; see Appendix C
xOffset	[in]x offset for the designated card side

Note: Valid offset values are 0 to 48.

Returns: error code (see)

Sample:

```
short xOffset = 27;
short alarm = job.Device.SetXOffset(SideEnum.Back, xOffset);
```

SetYOffset

Description: Assigns the Y offset for the specified side of the card.

Syntax:

```
short SetYOffset (
                SideEnum      side,
                short          yOffset )
```

Parameters:

side	[in]defines the side of the card; see Appendix C
yOffset	[in]y offset for the designated card side

Note: Valid offset values are 0 to 48.

Returns: error code (see)

Sample:

```
short yOffset = 27;
short alarm = job.Device.SetYOffset(SideEnum.Back, yOffset);
```

UpgradeEthernetFirmware

Description: The Printer uses an Ethernet Board which requires its own firmware. This SDK function updates the firmware on this Ethernet Board.

Syntax: short UpgradeEthernetFirmware (string ethernetFirmwareFile)

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

Note: Firmware file is a binary file.

Returns: error code (see)

Sample:

```
string ethernetFirmwareFile = "c:\\FirmwareFolder\\  
firmwareFile.bin";  
  
short alarm = job.Device.UpgradeEthernetFirmware(  
    ethernetFirmwareFile );
```

UpgradeFirmware

Description: This function updates the firmware in the printer's main board. The main board's firmware is responsible for the printer's functionality and features.

Syntax: short UpgradeFirmware (string firmwareFile)

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

Note: Firmware file is a binary file.

Returns: error code (see)

Sample:

```
string firmwareFile = "c:\\FirmwareFolder\\firmwareFile.bin";  
short alarm = job.Device.UpgradeFirmware( firmwareFile );
```


Deprecated Commands

Methods

ByteArrayToVariantArray	76
BytePtrToVariantArray	77
IntArrayToVariantArray	78
LongArrayToVariantArray	79
VariantArrayToByteArray	80
VariantArrayToIntArray	81
VariantArrayToLongArray	82



Note • All functions in this section are deprecated and should not be used.

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:

```
byte[] arrayToBeConverted; // previously instantiated and populated array           //
    containing data to be converted
object varArray = null;    // array to hold the converted data
job.Utilities.ByteArrayToVariantArray(arrayToBeConverted, out varArray);
```

BytePtrToVariantArray

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

Syntax:

```
void job.Utilities.BytePtrToVariantArray (
    ref byte      bytePtr
    int byte      Count
    out object     varArray )
```

Parameters:

bytePtr	[in] pointer to byte array
byteCount	[in] number of bytes in the byte array
	varArray[out] variant array created from byte ptr

Returns: nothing

Sample:

```
byte ptr; //address of byte array
int count = sizeof byte array;
object varArray = null;
```

```
job.Utilities.BytePtrToVariantArray(ref ptr, count, out varArray);
```

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:

```
int[] intArray; //integer array to be converted
object varArray; //variant array to hold converted values

job.Utilities.IntArrayToVariantArray((object)intArray, out varArray);
```

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:

```
long[] lgArray; //long array to be converted
object varArray; //variant array to hold converted values

job.Utilities.LongArrayToVariantArray((object)lgArray, out varArray);
```

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:

```
object varArray; //variant array to be converted
byte[] byteArray; //byte array to hold converted values

object temp = (object) byteArray;
job.Utilities.VariantArrayToByteArray(varArray, out temp);
```

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:

```
object varArray; //variant array to be converted
int[] intArray; //integer array to hold converted values

object temp = (object) intArray;
job.Utilities.VariantArrayToIntArray(varArray, out temp);
```

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:

```
object varArray; //variant array to be converted
long[] lgArray;  //long array to hold converted values

object temp = (object) lgArray;
job.Utilities.VariantArrayToLongArray(varArray, out temp);
```


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 ZXP SDK for Zebra ZXP Series 1 and ZXP Series 3 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 ZXP-1 or ZXP-3 printer and captured by the ZXP SDK functions. They are typically mechanical card movement and ribbon alerts. Alarms are independent of ZXP jobs and indicate if it is safe to proceed to the next job. They are returned as numbers.

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

if ( alarmValue != 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

Error Codes

03002	Printer busy
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

Error Codes

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

Error Codes

07005	Printhead temperature too high
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

Error Codes

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

Error Codes

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

Error Codes

17030	Invalid bottom laminate
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

Error Codes

65018	Device in session with another host
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	Wi-Fi not available
65025	Invalid media for requested operation
65026	Requested operation timed out



ZXP SDK Enumerations

Job Enums

ATMModeEnum

- ATM
- Auto
- Feeder

BoolTypeEnum

- False_BT
- True_BT
- NoChange

CardDirectionEnum

- Forward
- Reverse

ConnectionTypeEnum

- All
- Ethernet
- USB

DataSourceEnum

- NoData
- Track1Data
- Track2Data
- Track3Data

DestinationTypeEnum

- Eject
- Hold

FeederSourceEnum

- ATMSlot
- CardFeeder
- Internal

GraphicTypeEnum

- BMP
- NA

MagCoercivityEnum

- HighCo
- LowCo

MagEncodingTypeEnum

ISO
JIS

MonoConvTypeEnum

MonoBarcode
MonoDiffusion
MonoHalftone
MonoText

OCODisplayModeEnum

Blink
Normal
Scroll

PrintTypeEnum

Color
GrayDye
MonoK
MonoKHigh
MonoWhite
MonoWhiteHigh
Overlay
WhiteResin

SideEnum

Back
Front

SmartCardTypeEnum

Contact
Contactless
None

For ZXP Series 3C only

```
public enum PrintOptimizationModeEnum
{
    Quality = 0,
    Speed = 1,
}
```