ZXP Series 1
ZXP Series 3

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**
- 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
<table>
<thead>
<tr>
<th>Job Control</th>
<th>33</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>.33</td>
</tr>
<tr>
<td>Method List</td>
<td>.33</td>
</tr>
<tr>
<td>Methods</td>
<td>.34</td>
</tr>
<tr>
<td>GetBlackIntensity</td>
<td>.34</td>
</tr>
<tr>
<td>GetCyanIntensity</td>
<td>.35</td>
</tr>
<tr>
<td>GetHalfPanelOffset</td>
<td>.36</td>
</tr>
<tr>
<td>GetMagentaIntensity</td>
<td>.37</td>
</tr>
<tr>
<td>GetMonoConvType</td>
<td>.38</td>
</tr>
<tr>
<td>GetOverlayIntensity</td>
<td>.39</td>
</tr>
<tr>
<td>GetYellowIntensity</td>
<td>.40</td>
</tr>
<tr>
<td>SetBlackIntensity</td>
<td>.41</td>
</tr>
<tr>
<td>SetCyanIntensity</td>
<td>.42</td>
</tr>
<tr>
<td>SetHalfPanelOffset</td>
<td>.43</td>
</tr>
<tr>
<td>SetMagentaIntensity</td>
<td>.44</td>
</tr>
<tr>
<td>SetMonoConvType</td>
<td>.45</td>
</tr>
<tr>
<td>SetOverlayIntensity</td>
<td>.46</td>
</tr>
<tr>
<td>SetYellowIntensity</td>
<td>.47</td>
</tr>
<tr>
<td>SmartCardConfiguration</td>
<td>.48</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device</th>
<th>49</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>.49</td>
</tr>
<tr>
<td>Method List</td>
<td>.49</td>
</tr>
<tr>
<td>Methods</td>
<td>.50</td>
</tr>
<tr>
<td>BuildOCPMessage</td>
<td>.50</td>
</tr>
<tr>
<td>ClearOCPMessage</td>
<td>.51</td>
</tr>
<tr>
<td>DisplayOCPMessage</td>
<td>.52</td>
</tr>
<tr>
<td>GetConfiguration</td>
<td>.53</td>
</tr>
<tr>
<td>GetDeviceInfo</td>
<td>.54</td>
</tr>
<tr>
<td>GetDisplayedOCPMessage</td>
<td>.55</td>
</tr>
<tr>
<td>GetErrorCount</td>
<td>.56</td>
</tr>
<tr>
<td>GetMagneticEncoderConfiguration</td>
<td>.57</td>
</tr>
<tr>
<td>GetNetworkParams</td>
<td>.58</td>
</tr>
<tr>
<td>GetPrinterStatus</td>
<td>.59</td>
</tr>
<tr>
<td>GetRibbonParams</td>
<td>.60</td>
</tr>
<tr>
<td>GetSensorStates</td>
<td>.61</td>
</tr>
<tr>
<td>GetSensorValues</td>
<td>.62</td>
</tr>
<tr>
<td>GetSmartCardConfiguration</td>
<td>.63</td>
</tr>
<tr>
<td>GetStatusMessageString</td>
<td>.64</td>
</tr>
<tr>
<td>GetTotalCardCount</td>
<td>.65</td>
</tr>
<tr>
<td>GetXOffset</td>
<td>.66</td>
</tr>
<tr>
<td>GetYOffset</td>
<td>.67</td>
</tr>
<tr>
<td>MoveCard</td>
<td>.68</td>
</tr>
<tr>
<td>SendCommand</td>
<td>.69</td>
</tr>
<tr>
<td>SetNetworkParams</td>
<td>.70</td>
</tr>
<tr>
<td>SetXOffset</td>
<td>.71</td>
</tr>
<tr>
<td>SetYOffset</td>
<td>.72</td>
</tr>
<tr>
<td>UpgradeEthernetFirmware</td>
<td>.73</td>
</tr>
<tr>
<td>UpgradeFirmware</td>
<td>.74</td>
</tr>
</tbody>
</table>
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. Send 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.
Properties

Boolean
IsOpen (read only) indicates device connection status: returns true if open, false if closed
Float
EthernetOpenTimeout sets/get 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 {
    object objJobList = null;
    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:**
```csharp
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:**
```csharp
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 );
Job

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:**
```csharp
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.

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

//Ethernet Connection:
```csharp
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:
```csharp
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:
```csharp
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:

```csharp
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.

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

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

### ReadMagData

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

**Syntax:**
```csharp
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);
}
```

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);
}
```

//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:**
```c
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:**
```c
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 get/set 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

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetBlackIntensity</td>
<td>34</td>
</tr>
<tr>
<td>GetCyanIntensity</td>
<td>35</td>
</tr>
<tr>
<td>GetHalfPanelOffset</td>
<td>36</td>
</tr>
<tr>
<td>GetMonoConvType</td>
<td>38</td>
</tr>
<tr>
<td>GetOverlayIntensity</td>
<td>39</td>
</tr>
<tr>
<td>GetYellowIntensity</td>
<td>40</td>
</tr>
<tr>
<td>SetBlackIntensity</td>
<td>41</td>
</tr>
<tr>
<td>SetCyanIntensity</td>
<td>42</td>
</tr>
<tr>
<td>SetHalfPanelOffset</td>
<td>43</td>
</tr>
<tr>
<td>SetMonoConvType</td>
<td>45</td>
</tr>
<tr>
<td>SetOverlayIntensity</td>
<td>46</td>
</tr>
<tr>
<td>SetYellowIntensity</td>
<td>47</td>
</tr>
<tr>
<td>SmartCardConfiguration</td>
<td>48</td>
</tr>
</tbody>
</table>
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: 
```c
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: 
```c
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: 
```c
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: 
```c
short value = 0;
jobj.JobControl.GetOverlayIntensity(SideEnum.Front, out value)
```
GetYellowIntensity

**Description:** Returns the current yellow intensity value.

**Syntax:**
```c
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:**
```c
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**:
```c
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**:
```c
short value = 10;
job.JobControl.SetCyanIntensity(SideEnum.Back, value);
```
SetHalfPanelOffset

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

**Syntax:**
```csharp
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:**
```csharp
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:**
```csharp
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:**
```csharp
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:**
```java
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

BuildOCPMessage

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

Syntax: void BuildOCPMessage (
        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.BuildOCPMessage(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();
DisplayOCPMessage

**Description:** Displays the message created by BuildOCPMessage on the printer’s OCP.

**Syntax:**
```
short DisplayOCPMessage()
```

**Parameters:** none

**Returns:** error code (see )

**Sample:**
```
short alarm = job.Device.DisplayOCPMessage();
```
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:**
```csharp
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:**
```csharp
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 dhcp 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:

```csharp
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:
```csharp
string status = string.Empty;
int error = 0;
int jobsPending = 0;
int jobsActive = 0;
int jobsComplete = 0;
int jobErrors = 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:**
```c
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:**
```c
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:**

```c
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:**

```c
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 etherenFirmwareFile = "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.
**Deprecated Commands**

---

**ByteArrayToVariantArray**

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

**Syntax:**
```csharp
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:**
```csharp
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);
Deprecated Commands

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:**
```csharp
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:**
```csharp
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:**
```csharp
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:**
```csharp
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:**
```java
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:**
```csharp
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.

```java
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.

```java
short alarmValue = SDK Function ( ... )
if ( alarmValue != 0 )
    errMsg = job.Device.GetStatusMessageString(alarmValue);
else
    proceed to next job
```
## Error Codes and Descriptions

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

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>03002</td>
<td>Printer busy</td>
</tr>
<tr>
<td>03003</td>
<td>Invalid ZMC command</td>
</tr>
<tr>
<td>03004</td>
<td>Invalid ZMC sub-command</td>
</tr>
<tr>
<td>03005</td>
<td>Invalid ZMC parameter (1)</td>
</tr>
<tr>
<td>03006</td>
<td>Invalid ZMC parameter (2)</td>
</tr>
<tr>
<td>03007</td>
<td>Invalid ZMC parameter (3)</td>
</tr>
<tr>
<td>03008</td>
<td>Command processing error</td>
</tr>
<tr>
<td>03009</td>
<td>Response too large for host</td>
</tr>
<tr>
<td>03010</td>
<td>Host write occurred when host read expected</td>
</tr>
<tr>
<td>03011</td>
<td>Host read occurred when host write expected</td>
</tr>
<tr>
<td>03012</td>
<td>Data less than specified in header</td>
</tr>
<tr>
<td>03013</td>
<td>Data more than specified in header</td>
</tr>
<tr>
<td>03014</td>
<td>Communication synch error</td>
</tr>
<tr>
<td>03015</td>
<td>End Action error</td>
</tr>
<tr>
<td>03016</td>
<td>Cancel Action error</td>
</tr>
<tr>
<td>03017</td>
<td>No Start Action</td>
</tr>
<tr>
<td>03018</td>
<td>Start Action already called</td>
</tr>
<tr>
<td>03019</td>
<td>Job data error</td>
</tr>
<tr>
<td>03020</td>
<td>Memory-pool allocation error</td>
</tr>
<tr>
<td>03021</td>
<td>XML parse error</td>
</tr>
<tr>
<td>03022</td>
<td>Invalid payload length</td>
</tr>
<tr>
<td>03023</td>
<td>HMAC missing</td>
</tr>
<tr>
<td>03024</td>
<td>Invalid payload content</td>
</tr>
<tr>
<td>03025</td>
<td>Device reservation failed</td>
</tr>
</tbody>
</table>

**04000 - 04999  media errors (card, laminate, retransfer film, paper, etc)**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>04001</td>
<td>Out of cards</td>
</tr>
<tr>
<td>04002</td>
<td>Invalid card type</td>
</tr>
<tr>
<td>04003</td>
<td>Card jam</td>
</tr>
<tr>
<td>04004</td>
<td>Reserved</td>
</tr>
<tr>
<td>04005</td>
<td>Reserved</td>
</tr>
<tr>
<td>04006</td>
<td>Reserved</td>
</tr>
<tr>
<td>04007</td>
<td>Reserved</td>
</tr>
<tr>
<td>04008</td>
<td>Reserved</td>
</tr>
<tr>
<td>04009</td>
<td>Reserved</td>
</tr>
<tr>
<td>04010</td>
<td>Out of retransfer media</td>
</tr>
<tr>
<td>04011</td>
<td>Invalid retransfer media</td>
</tr>
<tr>
<td>04012</td>
<td>Retransfer media jam</td>
</tr>
<tr>
<td>04013</td>
<td>Retransfer media motion error</td>
</tr>
<tr>
<td>04014</td>
<td>Card not detected</td>
</tr>
<tr>
<td>04015</td>
<td>Insert card timeout</td>
</tr>
<tr>
<td>04016</td>
<td>Card feeder is empty</td>
</tr>
<tr>
<td>04017</td>
<td>Invalid retransfer media</td>
</tr>
</tbody>
</table>
### 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
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>07005</td>
<td>Printhead temperature too high</td>
</tr>
<tr>
<td>07006</td>
<td>Printhead temperature too low</td>
</tr>
<tr>
<td>07007</td>
<td>Protocol error</td>
</tr>
<tr>
<td>07008</td>
<td>Door open</td>
</tr>
<tr>
<td>07009</td>
<td>Invalid EP script</td>
</tr>
<tr>
<td>07010</td>
<td>Printhead CAM home error</td>
</tr>
<tr>
<td>07011</td>
<td>Transfer rollers temperature too high</td>
</tr>
<tr>
<td>07012</td>
<td>Transfer rollers temperature too low</td>
</tr>
<tr>
<td>07013</td>
<td>Motor voltage range error</td>
</tr>
<tr>
<td>07014</td>
<td>EP script processing error</td>
</tr>
<tr>
<td>07015</td>
<td>Mag retrace error</td>
</tr>
<tr>
<td>07016</td>
<td>Card transfer error</td>
</tr>
<tr>
<td>07017</td>
<td>Card reject error</td>
</tr>
<tr>
<td>07018</td>
<td>SmartCard positioning error</td>
</tr>
<tr>
<td>07019</td>
<td>EP script content error</td>
</tr>
<tr>
<td>07020</td>
<td>EP script transmission error</td>
</tr>
<tr>
<td>07021</td>
<td>Print path initialization error</td>
</tr>
<tr>
<td>07022</td>
<td>Flipper initialization error</td>
</tr>
<tr>
<td>07023</td>
<td>SmartCard cam error</td>
</tr>
<tr>
<td>07024</td>
<td>Options module card jam</td>
</tr>
<tr>
<td>07025</td>
<td>Print path card jam</td>
</tr>
<tr>
<td>07026</td>
<td>Flipper card jam</td>
</tr>
<tr>
<td>07027</td>
<td>Media drawer open</td>
</tr>
<tr>
<td>07028</td>
<td>Feeder door open</td>
</tr>
<tr>
<td>07029</td>
<td>Flipper move failure</td>
</tr>
<tr>
<td>07030</td>
<td>Reserved</td>
</tr>
<tr>
<td>07031</td>
<td>Reserved</td>
</tr>
<tr>
<td>07032</td>
<td>ATM card jam</td>
</tr>
<tr>
<td>07033</td>
<td>Reserved</td>
</tr>
<tr>
<td>07034</td>
<td>Reject bin full</td>
</tr>
<tr>
<td>07035</td>
<td>Magnetic encoder card jam</td>
</tr>
<tr>
<td>07036</td>
<td>Print path card jam</td>
</tr>
<tr>
<td>07037</td>
<td>Print synchronization card jam</td>
</tr>
<tr>
<td>07038</td>
<td>Print entrance card jam</td>
</tr>
<tr>
<td>07039</td>
<td>Print exit card jam</td>
</tr>
<tr>
<td>07040</td>
<td>Flipper initialization error</td>
</tr>
<tr>
<td>07041</td>
<td>Printhead initialization error</td>
</tr>
<tr>
<td>07042</td>
<td>Print path initialization error</td>
</tr>
<tr>
<td>07043</td>
<td>Options module initialization error</td>
</tr>
<tr>
<td>07044</td>
<td>SmartCard initialization error</td>
</tr>
</tbody>
</table>

### OCP errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>08001</td>
<td>Unresponsive</td>
</tr>
<tr>
<td>08002</td>
<td>Transmit error</td>
</tr>
</tbody>
</table>
## Error Codes

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

### 15000 - 15999 media authentication board (MAB) errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15001</td>
<td>Control board missing</td>
</tr>
<tr>
<td>15002</td>
<td>Bootloader mode - firmware reload may be required</td>
</tr>
</tbody>
</table>

### 16000 - 16999 security-related errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16001</td>
<td>Invalid passkey</td>
</tr>
<tr>
<td>16002</td>
<td>Invalid crypto key</td>
</tr>
<tr>
<td>16003</td>
<td>Authentication failed</td>
</tr>
<tr>
<td>16004</td>
<td>Invalid printer data</td>
</tr>
<tr>
<td>16005</td>
<td>Invalid HMAC</td>
</tr>
<tr>
<td>16006</td>
<td>Unsupported action</td>
</tr>
</tbody>
</table>

### 17000 - 17499 laminator board faults

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>17001</td>
<td>Missing laminator</td>
</tr>
<tr>
<td>17002</td>
<td>Initialization error</td>
</tr>
<tr>
<td>17003</td>
<td>Unknown error</td>
</tr>
<tr>
<td>17004</td>
<td>Media authentication board missing</td>
</tr>
<tr>
<td>17005</td>
<td>Top laminate feed error</td>
</tr>
<tr>
<td>17006</td>
<td>Bottom laminate feed error</td>
</tr>
<tr>
<td>17007</td>
<td>Top laminate registration error</td>
</tr>
<tr>
<td>17008</td>
<td>Staging error</td>
</tr>
<tr>
<td>17009</td>
<td>Early card jam</td>
</tr>
<tr>
<td>17010</td>
<td>Mid card jam</td>
</tr>
<tr>
<td>17011</td>
<td>Late card jam</td>
</tr>
<tr>
<td>17012</td>
<td>Poll timeout</td>
</tr>
<tr>
<td>17013</td>
<td>Top heater error - power off printer and correct problem</td>
</tr>
<tr>
<td>17014</td>
<td>Bottom heater error - power off printer and correct problem</td>
</tr>
<tr>
<td>17015</td>
<td>Top heater over temperature - power off printer and correct problem</td>
</tr>
<tr>
<td>17016</td>
<td>Bottom heater over temperature - power off printer and correct problem</td>
</tr>
<tr>
<td>17017</td>
<td>Top cutter stall - power off printer and correct problem</td>
</tr>
<tr>
<td>17018</td>
<td>Bottom cutter stall - power off printer and correct problem</td>
</tr>
<tr>
<td>17019</td>
<td>Top cutter failure - power off printer and correct problem</td>
</tr>
<tr>
<td>17020</td>
<td>Bottom cutter failure - power off printer and correct problem</td>
</tr>
<tr>
<td>17021</td>
<td>Top sensor failure - Power off printer and correct problem</td>
</tr>
<tr>
<td>17022</td>
<td>Bottom sensor failure - Power off printer and correct problem</td>
</tr>
<tr>
<td>17023</td>
<td>Fan failure - Power off printer and correct problem</td>
</tr>
<tr>
<td>17024</td>
<td>EEPROM corrupt</td>
</tr>
<tr>
<td>17025</td>
<td>Reserved</td>
</tr>
<tr>
<td>17026</td>
<td>Out of top and bottom laminate</td>
</tr>
<tr>
<td>17027</td>
<td>Out of top laminate</td>
</tr>
<tr>
<td>17028</td>
<td>Out of bottom laminate</td>
</tr>
<tr>
<td>17029</td>
<td>Invalid top laminate</td>
</tr>
</tbody>
</table>
## Error Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>17030</td>
<td>Invalid bottom laminate</td>
</tr>
<tr>
<td>17031</td>
<td>Bottom laminate registration error</td>
</tr>
<tr>
<td>17032</td>
<td>Remove top laminate</td>
</tr>
<tr>
<td>17033</td>
<td>Remove bottom laminate</td>
</tr>
<tr>
<td>17034</td>
<td>Remove top and bottom laminate</td>
</tr>
<tr>
<td>17035</td>
<td>Reserved</td>
</tr>
<tr>
<td>17036</td>
<td>Reserved</td>
</tr>
<tr>
<td>17037</td>
<td>Reserved</td>
</tr>
<tr>
<td>17038</td>
<td>Door open</td>
</tr>
<tr>
<td>17039</td>
<td>Reserved</td>
</tr>
<tr>
<td>17040</td>
<td>Initializing</td>
</tr>
<tr>
<td>17041</td>
<td>Bootloader mode - firmware reload may be required</td>
</tr>
<tr>
<td>17042</td>
<td>MAB Bootloader mode - firmware reload may be required</td>
</tr>
</tbody>
</table>

### 18000 - 18999  wired network (ethernet) errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>18001</td>
<td>Open failed</td>
</tr>
</tbody>
</table>

### 19000 - 19999  wireless network (WiFi) errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>19001</td>
<td>Open failed</td>
</tr>
<tr>
<td>19002</td>
<td>Access point missing</td>
</tr>
<tr>
<td>19003</td>
<td>Link lost</td>
</tr>
<tr>
<td>19004</td>
<td>Incompatible configuration</td>
</tr>
<tr>
<td>19005</td>
<td>Association failed</td>
</tr>
<tr>
<td>19006</td>
<td>Connection failed</td>
</tr>
</tbody>
</table>

### 65000 SDK+ errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>65001</td>
<td>Device not open</td>
</tr>
<tr>
<td>65002</td>
<td>Device already open</td>
</tr>
<tr>
<td>65003</td>
<td>Device not available</td>
</tr>
<tr>
<td>65004</td>
<td>Device not responding</td>
</tr>
<tr>
<td>65005</td>
<td>Bad ZMC response signature</td>
</tr>
<tr>
<td>65006</td>
<td>Bad ZMC Command echo</td>
</tr>
<tr>
<td>65007</td>
<td>Insufficient data received from device</td>
</tr>
<tr>
<td>65008</td>
<td>Invalid argument</td>
</tr>
<tr>
<td>65009</td>
<td>Path to XML element not found</td>
</tr>
<tr>
<td>65010</td>
<td>Parse error</td>
</tr>
<tr>
<td>65011</td>
<td>Empty/Invalid Data Structure</td>
</tr>
<tr>
<td>65012</td>
<td>Buffer overflow</td>
</tr>
<tr>
<td>65013</td>
<td>SmartCard Encoder not available</td>
</tr>
<tr>
<td>65014</td>
<td>Encryption error</td>
</tr>
<tr>
<td>65015</td>
<td>Job status error</td>
</tr>
<tr>
<td>65016</td>
<td>Dual sided printing not supported</td>
</tr>
<tr>
<td>65017</td>
<td>Unable to obtain exclusive access to device</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>65018</td>
<td>Device in session with another host</td>
</tr>
<tr>
<td>65019</td>
<td>Invalid device for requested operation</td>
</tr>
<tr>
<td>65020</td>
<td>Passphrase or security key required for requested operation</td>
</tr>
<tr>
<td>65021</td>
<td>Memory allocation error</td>
</tr>
<tr>
<td>65022</td>
<td>No devices found</td>
</tr>
<tr>
<td>65023</td>
<td>Disconnect error</td>
</tr>
<tr>
<td>65024</td>
<td>Wi-Fi not available</td>
</tr>
<tr>
<td>65025</td>
<td>Invalid media for requested operation</td>
</tr>
<tr>
<td>65026</td>
<td>Requested operation timed out</td>
</tr>
</tbody>
</table>
**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,
}