Zebra Card Printer

Encoding over Ethernet Software Developer Reference Manual

ZEBRA

P1103866-001

Copyright

© **2018 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

Encoding over Ethernet

Introduction

This manual contains information for software developers to write applications for Zebra card printers which require smartcard encoding via an Ethernet connection.

The purpose of the Encoding over Ethernet (EoE) SDK—ZBRSXBridge.dll—is to create the required communication interface between a software application and the smartcard module in a Zebra card printer.

Required Skills

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

SDK Elements

ZBRSXBridge.dll

- Provides the actual EoE API.
- It is written in Microsoft's C language
- Available as a 32 bit & 64 bit dynamic link library

Sxuptp.dll (Silex)

- Provides the interface to the required device driver
- It is written in Microsoft C language
- Available as a 32- and 64-bit dynamic link library

Sxuptp.sys (Silex)

- Device driver
- Required to be installed in the OS prior to using the SDK
- Available as 32- and 64-bit

Each of the DLLs should be placed in the same directory as the application which uses them.

The device driver should be installed using the supplied installation program.

Overview

Encoding over Ethernet is a two-step process: A virtual USB connection is created between a host (PC) and the smartcard module within a Zebra card printer over Ethernet. Once this connection is established, an additional connection is created between a software application and the smartcard to be encoded which remains active during the entire encoding process.

For UHF smartcards to be encoded, the Zebra SmartCard SDK and ZBRSCReader.dll will be required to communicate with the UHF cards. For encoding PC/SC compliant smartcards, the PC/SC SDK available within the operating system in use will be needed. Each of these SDKs is used following the establishment of the virtual USB connection.

ZBRSXBridge

ZBRSXClose

Description:	Closes the SDK.
Syntax:	int ZBRSXClose(
	out int errorCode)
Parameters:	errorCode [out] error code; see Appendix A
Return:	0 = function failed. See error code for details
	1 = function succeeded

Example:

```
int errorCode = 0;
int result = ZBRSXClose(out errorCode);
```

ZBRSXDiscover

Description:	Locates the smartcard module within the ZXP-Series card printer.
Syntax:	int ZBRSXDiscover(
	object IPAddress,
	out object RetDevice,
	out int errorCode)
Parameters:	IPAddress [in]IP Address of printer. String array cast to an object
	RetDevice [out]IP Address of the printer containing the smartcard module. String array returned as an object
	errorCode [out]error code; see Appendix A
Return:	0 = function failed. See error code for details
	1 = function succeeded

Example:

```
int errorCode = 0;
string[] ipAddress = new string[1];
object RetDevice = null;
ipAddres[0] = "10.1.24.150"
int result = ZBRSXDiscover(ipAddress, out RetDevice, out errorCode);
if(result == 1) //success
       {
              string[] device = (string[])RetDevice;
              string ipAddress = device[0];
       }
```

ZBRSXUSBEnumEx

Description: Enumerates the smartcard module located in the printer.

Syntax: int ZBRSXUSBEnumEx(

object IPAddress,

out object usbDevices,

out int errorCode)

Parameters: IPAddress [in] IP Address of printer. Object returned from ZBRSXDicover.

usbDevices [out] Smartcard module identifier. String array returned as an object: should be a single-element string array.

errorCode [out] error code; see Appendix A

Return: 0 = function failed. See error code for details

1 = function succeeded

Example:

```
int errorCode = 0;
object ipAddress;
//object returned from ZBRSXDiscover (2nd param of function)
object usbDevices = null;
int result = ZBRSXUSBEnumEx(ipAddress, out usbDevices, out errorCode);
if(result == 1 && errorCode == 0) //success
       {
              string[] devices = (string[])usbDevice;
              string scModuleID = devices[0];
       }
```

ZBRSXConnect

Description: Opens a “virtual USB” connection to the smartcard module located in the printer.

Syntax: int ZBRSXConnect(

string scModuleID,

bool encrypt,

out int errorCode)

Parameters: scModuleID [in]Smartcard module identifier returned from ZBRSXUSBEnumEx.

encrypt [in]Flag indicating whether or not the connection should be encrypted.

false = do not encrypt connection

true = encrypt connection

errorCode [out]error code; see Appendix A

Return: 0 = function failed. See error code for details

1 = function succeeded

Example:

```
int errorCode = 0;
scModuleID; //smartcard module identifier returned by ZBRSXUSBEnumEx
bool encrypt = true; //encrypt the connection
int result = ZBRSXConnect(scModuleID, encrypt, out errorCode);
if(result == 1 && errorCode == 0) //success
       {
              //virtual connection succeeded
       }
```

ZBRSXDisconnect

Description:	Closes a “virtual USB” connection to the smartcard module located in the printer.
Syntax:	int ZBRSXDisconnect(
	string scModuleID,
	out int errorCode)
Parameters:	scModuleID [in]Smartcard module identifier returned from ZBRSXUSBEnumEx.
	errorCode [out]error code; see Appendix A
Return:	0 = function failed. See error code for details
	1 = function succeeded

Example:

```
int errorCode = 0;
scModuleID; //smartcard module identifier returned by ZBRSXUSBEnumEx
int result = ZBRSXDisconnect(scModuleID, out errorCode);
if(result == 1 && errorCode == 0)
        {
                //virtual disconnection succeeded
        }
```

ZBRSXGetStatus

Description:	Returns the current state of a “virtual USB” connection.
Syntax:	int ZBRSXGetStatus(
	string scModuleID,
	out int status,
	out int errorCode)
Parameters:	scModuleID [in]Smartcard module identifier returned from ZBRSXUSBEnumEx.
	status [out]current status of the virtual usb connection:
	1 = not connected
	2 = connected
	3 = in use by another program
	errorCode [out]error code; see Appendix A
Return:	0 = function failed. See error code for details
	1 = function succeeded

Example:

```
int errorCode = 0;
int status = 0;
scModuleID; //smartcard module identifier returned by ZBRSXUSBEnumEx
int result = ZBRSXGetStatus(scModuleID, out status, out errorCode);
if(result == 1 && errorCode == 0)
        {
                //function call succeeded - check the status
                //variable for current state
                //of the connection
        }
```

ZBRSXGetPCSCReaderNames

Description: Returns the enumerated names for the contact and contactless readers located in the printer.

Syntax: int ZBRSXGetPCSCReaderNames(

string scModuleID,

out object readerNames,

out int errorCode)

Parameters:	scModuleID	[in]Smartcard module identifier returned from ZBRSXUSBEnumEx.
	readerNames	[out]String array containing the contact and contactless reader names. String array returned as an object: should be a two-element string array.
	errorCode	[out]error code; see Appendix A

Return: 0 = function failed. See error code for details

1 = function succeeded

Example:

```
int errorCode = 0;
scModuleID; //smartcard module identifier returned by ZBRSXUSBEnumEx
object readerNames = null;
int result = ZBRSXGetPCSCReaderNames(scModuleID, out readerNames,
       out errorCode);
if(result == 1 && errorCode == 0) //success
       {
              string[] devices = (string[])readerNames;
              string contactName = devices[0];
              string contactlessName = devices[1];
       }
```

Using ZBRSXBridge

Introduction

This chapter demonstrates how to use the ZBRSXBridge.dll to create the required “virtual usb” connection, to retrieve the smartcard module ID, locate the enumerated smartcard reader names, and close the “virtual usb connection.”

Note: the enumerated smartcard reader names are required when performing smartcard encoding via the PC/SC SDK only. The UHF SDK has no such requirement.

DLL Loader

The library DLLs need to be copied to the application’s working directory. The file copy code should be run at application initialization.

```
string dllPath = path to where the libraries were installed

if ( !File.Exists ( “ZBRSXBridge.dll” )) {
       File.Copy ( dllPath  + “/ZBRSXBridge.dll”, “ZBRSXBridge.dll” );
}
if ( !File.Exists ( “Zbscfgsrv.dll” )) {
       File.Copy ( dllPath  + “/Zbscfgsrv.dll”, “Zbscfgsrv.dll” );
}
if ( !File.Exists ( “Sxuptp.dll” )) {
       File.Copy ( dllPath  + “/Sxuptp.dll”, “Sxuptp.dll” );
}
```

DLL Importer

An application that accesses the methods in the ZBRSXBridge library must provide a reference to the library and methods.

```
public class BridgeImport {

DllImport(“ZBRSXBridge.dll”, EntryPoint = “ZBRSXClose”, CharSet = CharSet.Auto,
       CallingConvention = CallingConvention.StdCall, SetLastError = true)]
public static extern int ZBRSXClose(out int errorValue);

[DllImport(“ZBRSXBridge.dll”, EntryPoint = “ZBRSXDiscover”, CharSet = CharSet.Auto,
       CallingConvention = CallingConvention.StdCall, SetLastError = true)]
public static extern int ZBRSXDiscover(object ipAddresses, out object retDevices, out int
errorValue);

[DllImport(“ZBRSXBridge.dll”, EntryPoint = “ZBRSXUSBEnumEx”, CharSet = CharSet.Auto,
       CallingConvention = CallingConvention.StdCall, SetLastError = true)]
public static extern int ZBRSXUSBEnumEx(object retDevices, out object deviceIDs, out int
errorValue);

[DllImport(“ZBRSXBridge.dll”, EntryPoint = “ZBRSXConnect”, CharSet = CharSet.Auto,
       CallingConvention = CallingConvention.StdCall, SetLastError = true)]
public static extern int ZBRSXConnect(string deviceID, bool encrypt, out int errorValue);

[DllImport(“ZBRSXBridge.dll”, EntryPoint = “ZBRSXDisconnect”, CharSet = CharSet.Auto,
       CallingConvention = CallingConvention.StdCall, SetLastError = true)]
public static extern int ZBRSXDisconnect(string deviceID, out int errorValue);

[DllImport(“ZBRSXBridge.dll”, EntryPoint = “ZBRSXGetStatus”, CharSet = CharSet.Auto,
       CallingConvention = CallingConvention.StdCall, SetLastError = true)]
public static extern int ZBRSXGetStatus(string deviceID, out int status, out int errorValue);

[DllImport(“ZBRSXBridge.dll”, EntryPoint = “ZBRSXGetPCSCReaderNames”, CharSet = CharSet.Auto,
       CallingConvention = CallingConvention.StdCall, SetLastError = true)]
public static extern int ZBRSXGetPCSCReaderNames(string deviceID, out object readerNames, out int
errorValue);

}
```

A class that needs to reference the ZBRSXBridge library methods could access then through inheritance as such:
class ExampleEoE : BridgeImport {

ZBRSXBridge Methods

ZBRSXCLOSE

Description:	Closes the ZBRSXBridge library
Syntax:	int ZBRSXClose (out int errorValue)
Parameters:	errorValue see Appendix A
Returns:	1 = function succeeded; 0 = function failed

Example:

```
int ret = ZBRSXClose ( out errorValue );
```

ZBRSXDiscover

Description:	discovers Ethernet connected systems available for USB Virtual connections base on a list of possible IP Addresses	
Syntax:	int ZBRSXDiscover (object ipAddresses, out object retDevices, out int errorValue)	
Parameters:	ipAddresses	list of IP Addresses to use for discovery
	retDevices	list of discovered IP Address supporting Virtual USB connections
	errorValue	see Appendix A
Returns:	1 = function succeeded; 0 = function failed	

Example:

```
string[] ipAddresses = {"10.1.24.10", "10.1.28.150", "10.1.30.155"};
object retDevices = null;
int ret = ZBRSXDiscover(ipAddresses, out retDevices, out errorValue);
```

ZBRSXUSBEnumEx

Description:	enumerates the discovered devices	
Syntax:	int ZBRSXUSBEnumEx (object retDevices, out object deviceIDs, out int errorValue)	
Parameters:	retDevices	list of IP Addresses generated by ZBRSXDiscover
	deviceIDs	list of device identifier strings used during the Virtual USB process
	errorValue	see Appendix A
Returns:	1 = function succeeded; 0 = function failed	

Example:

```
object deviceIDs = null;
int ret = ZBRSXUSBEnumEx ( retDevices, out deviceIDs, out errorValue );
string[] strDeviceIDs = (string[])deviceIDs;
```

ZBRSXConnect

Description:	establishes a Virtual USB connection based on a Device ID
Syntax:	int ZBRSXConnect (string deviceID, bool encrypt, out int retError);
Parameters:	deviceID identifies the device for the Virtual USB connection
	encrypt indicates if data sent over the Virtual USB cannel is to be encrypted
	errorValue see Appendix A
Returns:	1 = function succeeded; 0 = function failed

Example:

```
int status = 0;
if (ZBRSXGetStatus(deviceID, out status, out errorValue ) == 1) {
       if (status != 2 && status != 3) {
       ret = ZBRSXConnect(deviceID, false, out errorValue);
       }
}
```

ZBRSXDisconnect

Description:	closes the Virtual USB connection for the Device ID
Syntax:	int ZBRSXDisconnect (string deviceID, out int retError);
Parameters:	deviceID identifies the Virtual USB connection to close
	errorValue see Appendix A
Returns:	1 = function succeeded; 0 = function failed
Example:	int ret = ZBRSXDisconnect(deviceID, out errorValue);

ZBRSXGetStatus

Description:	gets connection status for a Device ID
Syntax:	int ZBRSXGetStatus (string deviceID, out int status, out int errorValue);
Parameters:	deviceID identifies the Virtual USB connection to check
	Status Virtual USB connection status
	1 = not connected
	2 = connected
	3 = in use by another application
	errorValue see Appendix A
Returns:	1 = function succeeded; 0 = function failed

Example:

```
int status = 0;
if ( ZBRSXGetStatus ( deviceID, out status, out errorValue ) == 1) {
       bool connected = (status == 2 || status == 3);
}
```

ZBRSXGetPCSCReaderName

Description: gets the smart card readers names for the Device ID

Syntax: int ZBRSXGetPCSCReaderNames (string deviceID, out object readerNames, out int errorValue);

Parameters:

deviceID	identifies the Virtual USB connection to check for smart card readers
readerNames	list of smart card readers

Returns: 1 = function succeeded; 0 = function failed

Example:

```
object readerNames = null;
if ( ZBRSXGetPCSCReaderNames(deviceID,
              out readerNames,
              out errorValue ) == 1) {
       string [] strReaderName = (string[])readerNames;
}
```

Example Code:

```
public bool Example() {

    bool passed = false;

    try {

        int errorValue = 0;

        // * String of IP Addresses for the discovery process
        string[] ipAddresses = {“192.168.0.15”, “10.1.24.150”, “10.1.30.155”};

        // * List of discovered IP Address that can be used for Virtual USB connections
        object retDevices = null;

        if (ZBRSXDiscover(ipAddresses, out retDevices, out errorValue) != 1) {
            throw new Exception(“Discovery Error: “ + errorValue.ToString());
        }

        // * List of Device IDs ( descriptions ) for the discovered devices
        // * Device IDs will be used for the Virtual USB methods
        object deviceIDs = null;

        if (ZBRSXUSBEnumEx(retDevices, out deviceIDs, out errorValue) != 1) {
            throw new Exception(“USB Enum Ex Error: “ + errorValue.ToString());
        }
        string[] strDeviceIDs = (string[])deviceIDs;
        string deviceID = strDeviceIDs[0];

        // * Determines if there is already a connection for the Device ID
        // * 1 = not connected; 2 = connected; 3 = already connected
        int status = 0;
        if (ZBRSXGetStatus(deviceID, out status, out errorValue ) != 1) {
            throw new Exception(“Get Status Error: “ + errorValue.ToString());
        }
        bool connected = (status == 2 || status == 3);

        // * Creates if Virtual USB connection
        bool encrypted = false;
        if (!connected) {
            if (ZBRSXConnect(deviceID, encrypted, out errorValue) != 1) {
                throw new Exception(“Connection Error: “ + errorValue.ToString());
            }
        }

        // * Next step is to get the smart card reader name
        object readerNames = null;
        if (ZBRSXGetPCSCReaderNames(deviceID, out readerNames, out errorValue ) != 1) {
            throw new Exception(“Get Reader Names Error: “ + errorValue.ToString());
        }
        string [] strReaderName = (string[])readerNames;

        // ***** PC/SC code goes here *****

        if (ZBRSXDisconnect(deviceID, out errorValue) != 1) {
            throw new Exception(“Disconnect Error: “ + errorValue.ToString());
        }

        if (ZBRSXClose(out errorValue) != 1) {
            throw new Exception(“Close Error: “ + errorValue.ToString());
        }

        passed = true;

    } catch(Exception ex) {
        this.errorDescr = ex.Message;
    }

    return passed;
}
```

Error Codes

This appendix lists the error codes, error messages, and possible causes for the errors that may appear when running applications created with this SDK.

Error Codes and Descriptions

CODE	DESCRIPTION
65001	Device not open
65002	Device already open
65003	Device not available
65004	Device not responding
65005	Bad ZMC response signature
65006	Bad ZMC Command echo
65007	Insufficient data received from device
65008	Invalid argument
65009	Path to XML element not found
65010	Parse error
65011	Empty/Invalid Data Structure
65012	Buffer overflow
65013	SmartCard Encoder not available
65014	Encryption error
65015	Job status error
65016	Dual sided printing not supported
65017	Unable to obtain exclusive access to device
65018	Device in session with another host
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