Zebra® Card Printer

Software Development Kit
Reference Manual

March 30, 2009.

980592-001 Beta 4

Card Printer Solutions

Copyright Notice

Trademarks

Contents

1 • Introduction 1
About This Manual 1
Required Skills 1
Zebra Card Printers 2
Communication Ports 2
SDK Elements 3
Printer 3
Graphics 3
GemCore 3
MIFARE 3
UHF 3
Installation 4
Card-Handling 5

2 • Printer Functions 7
Introduction 7
Required Skills 7
Zebra Card Printers 7
Communication Ports 7
Printer SDK Elements 8
Installation 8
Function List 9
SDK Specific Function 12
ZBRPRNGetSDKVer 12
ZBRPRNGetSDKVsn 13
Printer Driver Handle Functions 14
ZBRGetHandle 14
ZBRCloseHandle 15
Printer Command Functions 16
ZBRPRNSendCmd 16
ZBRPRNSendCommand 17
ZBRPRNSendCmdEx 18

ZBRPRNMultipleCmd 19
ZBRPRNPrintPrnFile 20
Status Functions 21
ZBRPRNClrErrStatusLn 21
ZBRPRNGetPrintCount 22
ZBRPRNGetPrinterSerialNumber 23
ZBRPRNGetPrinterSerialNumb 24
ZBRPRNGetPrinterOptions 25
ZBRPRNGetPrintHeadSerialNumber 26
ZBRPRNGetPrintHeadSerialNumb 27
ZBRPRNGetOpParam 28
ZBRPRNGetPrinterStatus 29
ZBRPRNGetSensorStatus 30
ZBRPRNIsPrinterReady 31
Cleaning Functions 32
ZBRPRNChkDueForCleaning 32
ZBRPRNStartCleaningSeq 33
ZBRPRNStartCleaningCardSeq 34
ZBRPRNGetCleaningParam 35
ZBRPRNSetCleaningParam 36
Printer Setup Functions 37
ZBRPRNResetPrinter 37
ZBRPRNSelfAdj 38
ZBRPRNGetChecksum 39
ZBRPRNSetCardFeedingMode 40
ZBRPRNSetPrintHeadResistance 41
ZBRPRNClrMediaPath 42
ZBRPRNImmediateParamSave 43
ZBRPRNSetRelativeXOffset 44
ZBRPRNSetRelativeYOffset 45
ZBRPRNSetStartPrintXOffset 46
ZBRPRNSetStartPrintYOffset 47
ZBRPRNSetStartPrintSideBOffset 48
ZBRPRNSetStartPrintSideBXOffset 49
ZBRPRNSetStartPrintSideBYOffset 50
Image Buffer Functions 51
ZBRPRNSetColorContrast 51
ZBRPRNSetContrastIntensityLvl 52
ZBRPRNSetHologramIntensity 53
ZBRPRNSetMonoContrast 54
ZBRPRNSetMonoIntensity 55
ZBRPRNClrSpecifiedBmp 56
ZBRPRNClrMonoImgBuf 57
ZBRPRNClrMonoImgBufs 58
ZBRPRNClrColorImgBufs 59
ZBRPRNClrColorImgBuf 60
ZBRPRNPrintMonoImgBuf 61
ZBRPRNPrintMonoImgBufEx 62
ZBRPRNPrintColorImgBuf 63
ZBRPRNPrintClearVarnish 64
ZBRPRNPrintVarnish 65
ZBRPRNPrintVarnishEx 66

ZBRPRNPrintHologramOverlay 67
ZBRPRNPrintCardPanel 68
ZBRPRNPrintMonoPanel 69
ZBRPRNWriteBox 70
ZBRPRNWriteBoxEx 71
ZBRPRNWriteText 72
ZBRPRNWriteTextEx 73
ZBRPRNSetEndOfPrint 74
Position Card Functions 75
ZBRPRNMovePrintReady 75
ZBRPRNReversePrintReady 76
ZBRPRNEjectCard 77
ZBRPRNFlipCard 78
ZBRPRNMoveCard 79
ZBRPRNMoveCardBkwd 80
ZBRPRNMoveCardFwd 81
ZBRPRNResync 82
Test Card Function 83
ZBRPRNPrintTestCard 83
Barcode Card Function 84
ZBRPRNWriteBarCode 84
Magnetic Encoder Functions 85
ZBRPRNSetEncodingDir 85
ZBRPRNSetTrkDensity 86
ZBRPRNResetMagEncoder 87
ZBRPRNSetEncoderCoercivity 88
ZBRPRNSetMagEncodingStd 89
ZBRPRNReadMag 90
ZBRPRNReadMagByTrk 91
ZBRPRNWriteMag 92
ZBRPRNWriteMagByTrk 93
ZBRPRNWriteMagPassThru 94

3 • Graphic Functions 97

Introduction 97
Required Skills 97
Zebra Card Printers 97
Communication Ports 97
SDK Elements 98
Installation 98
Function List 99
SDK Specific Function 100
ZBRGDIGetSDKVer 100
ZBRGDIGetSDKVsn 101
Initialization Functions 102
ZBRGDIInitGraphics 102
ZBRGDIInitGraphicsEx 103
ZBRGDIInitGraphicsFromPrintDlg 104
ZBRGDICloseGraphics 105
ZBRGDIClearGraphics 106
ZBRGDIStartPage 107
ZBRGDIEndPage 108

Print Functions 109
ZBRGDIPreviewGraphics 109
ZBRGDIPrintGraphics 110
ZBRGDIPrintGraphicsEx 111
ZBRGDIPrintFilePos 112
ZBRGDIPrintFileRect 113
ZBRGDIIsPrinterReady 114
Draw Functions 115
ZBRGDIDrawText 115
ZBRGDIDrawTextUnicode 116
ZBRGDIDrawTextEx 117
ZBRGDIDrawTextRect 118
ZBRGDIDrawTextRectEx 119
ZBRGDIDrawLine 120
ZBRGDIDrawImage 121
ZBRGDIDrawImagePos 122
ZBRGDIDrawImageRect 123
ZBRGDIDrawRectangle 124
ZBRGDIDrawEllipse 125
ZBRGDIDrawBarCode 126

4 • GemCore Functions 127

Introduction 127
Required Skills 127
Zebra Card Printers 127
Communication Ports 127
SDK Elements 128
Installation 128
Function List 129
SDK Specific Function 130
ZBRGCGetSDKVer 130
ZBRGCGetSDKVsn 131
Printer Functions 132
ZBRGetHandle 132
ZBRCloseHandle 133
ZBRGCStartCard 134
ZBRGCEndCard 135
ZBRGCEndCardEx 136
Card Specific Functions 137
ZBRGCCardPowerUp 137
ZBRGCCardPowerUpEx 138
ZBRGCCardPowerUpPPS 139
ZBRGCCardPowerDown 140
ZBRGCReaderPowerDown 141
ZBRGCExchangeData 142
ZBRGCExchangeAPDU 143
ZBRGCISOInput 144
ZBRGCISOOutput 145
ZBRGCCardStatus 146
Reader Specific Functions 147
ZBRGCSetCardType 147
ZBRGCDirectory 148

ZBRGCReadFirmwareVer 149
ZBRGCReadFirmwareVsn 150
ZBRGCGetOpMode 151
ZBRGCSetOpMode 152
ZBRGCGetTimeout 153
ZBRGCSetTimeout 154

5 • MIFARE Functions 155

Introduction 155
Required Skills 155
Zebra Card Printers 155
Communication Ports 155
MIFARE SDK Elements 156
Installation 156
Function List 157
DLL Function 159
ZBRGPMFGetSDKVer 159
ZBRGPMFSDKGetVer 160
Printer Functions 161
ZBRGetHandle 161
ZBRCloseHandle 162
ZBRGPMFStartCard 163
ZBRGPMFEndCard 164
ZBRGPMFEndCardEx 165
Card Functions 166
ZBRGPMF_LoadKey 166
ZBRGPMF_Authenticate 167
ZBRGPMF_Read 168
ZBRGPMF_Write 169
ZBRGPMF_SubtractValue 170
ZBRGPMF_AddValue 171
ZBRGPMF_Restore 172
ZBRGPMF_Transfer 173
Purse Card Functions 174
ZBRGPMF_B_CreatePurse 174
ZBRGPMF_B_ReadPurse 175
ZBRGPMF_B_DebitPurse 176
ZBRGPMF_B_CreditPurse 177
MAD Card Function 178
ZBRGPMF_MAD_ReadDataSector 178
Combined Card Functions 179
ZBRGPMF_C_Read 179
ZBRGPMF_C_Write 180
ZBRGPMF_C_CreateValueBlock 181
ZBRGPMF_C_ReadValue 182
ZBRGPMF_C_SubtractValue 183
ZBRGPMF_C_AddValue 184
ZBRGPMF_C_CopyValue 185
ZBRGPMF_C_SetAccessConditions 186
Reader Functions 187
ZBRGPMF_Reader_GetFirmware 187
ZBRGPMF_Reader_GetID 188

ZBRGPMF_Reader_GetModeAndGBPAddress 189
ZBRGPMF_Reader_SetMode 190
ZBRGPMF_Reader_ReadEEPROM 191
ZBRGPMF_Reader_WriteEEPROM 192
ZBRGPMF_Reader_GetParameters 193
ZBRGPMF_Reader_ActivateBuzzer 194
ZBRGPMF_Reader_ChangeMode. 195
ZBRGPMF_Reader_ControlLeds. 196
ZBRGPMF_Reader_OpenCaseDetection 197
ZBRGPMF_Reader_SetDelay 198
RF Functions. 199
ZBRGPMF_RF_Control 199
ZBRGPMF_RF_ChangeModulationType 200
ZBRGPMF_RF_ReadModulationType 201
14443A Functions 202
ZBRGPMF_ISO14443_3_A_RequestA 202
ZBRGPMF_ISO14443_3_A_Anticollision 203
ZBRGPMF_ISO14443_3_A_Select 204
ZBRGPMF_ISO14443_3_A_Halt 205
Combined 14443A Functions 206
ZBRGPMF_ISO14443_3_A_GetCard 206
ZBRGPMF_ISO14443_3_A_RequestAllSelectA 207
ZBRGPMF_ISO14443_3_A_GetCardA_T_CL. 208
ZBRGPMF_ISO14443_3_A_RequestAllSelectA_T_CL. 209
14443_4_A Functions. 210
ZBRGPMF_ISO14443_4_A_RequestForAnswerToSelect. 210
ZBRGPMF_ISO14443_4_A_ProtocolParameterSelection. 211
14443_4_A_B Functions. 212
ZBRGPMF_ISO14443_4_A_B_Exchange_T_CL 212
ZBRGPMF_ISO14443_4_A_B_Deselect. 213
ZBRGPMF_ISO14443_4_A_B_Poll_T_CL_Card_Removed. 214
ZBRGPMF_ISO14443_4_A_B_Mode15_GetStatus 215
ZBRGPMF_GEMCORECARDI_Card_Exchange 216
ZBRGPMF_GEMCORECARDI_Exchange_IFSD 217
ZBRGPMF_GEMCORECARDI_Exchange_PPS. 218
ZBRGPMF_GEMCORECARDI_PowerDown. 219
ZBRGPMF_GEMCORECARDI_PowerUp 220
ZBRGPMF_GEMCORECARDI_SetCurrentSAM. 221
ZBRGPMF_GEMCORECARDI_Status 222
14443B Functions. 223
ZBRGPMF_ISO14443_3_B_RequestB 223
ZBRGPMF_ISO14443_3_B_SlotMarker 224
ZBRGPMF_ISO14443_3_B_Attribute 225
ZBRGPMF_ISO14443_3_B_Halt. 226
Combined 14443B Function 227
ZBRGPMF_ISO14443_3_B_GetCard 227
Transparent Functions 228
ZBRGPMF_TransparentExchange. 228
ZBRGPMF_TransparentExchangeTimeout 229
ZBR_TL_SendReceive. 230
Structure Definitions 231

Access Conditions . . . 233
Data Block . . . 233
Sector Trailer . . . 233
MIFARE Key Management . . . 234
EEPROM Management . . . 235
Answer To Request, Type A (ATQA) . . . 236
Select Acknowledge (SAK) . . . 237
Glossary . . . 238

6 • UHF Functions . . . 239
Introduction . . . 239
Required Skills . . . 239
Zebra Card Printers . . . 239
Communication Ports . . . 239
SDK Elements . . . 240
Installation . . . 240
Function List . . . 241
ZBRUHFGetSDKVer . . . 242
ZBRGetHandle . . . 243
ZBRCloseHandle . . . 244
ZBRUHFStartCard . . . 245
ZBRUHFEndCard . . . 246
ZBRUHFEndCardEx . . . 247
ZBRUHFGetReaderVersions . . . 248
ZBRUHFGetCurrentRegion . . . 249
ZBRUHFGetAvailableRegions . . . 250
ZBRUHFSend . . . 251
ZBRUHFReceive . . . 252
ZBRUHFSendReceive . . . 253
ZBRUHFReadTagID . . . 254
ZBRUHFAppendChecksum . . . 255
ZBRUHFWriteTagID . . . 256
ZBRUHFLockTag . . . 257
ZBRUHFWriteTagPasswords . . . 258
ZBRUHFWriteTagData . . . 259
ZBRUHFReadTagData . . . 260
ZBRUHFUnlockTag . . . 261
ZBRUHFReset . . . 262

7 • Programming Examples . . . 263
Basic Card Printing and Magnetic Stripe Encoding . . . 264
Contact Smart Card . . . 267
MIFARE . . . 270
UHF . . . 275
Barcode . . . 277

Appendix A • Error Codes . . . 279
i -Series Specific Error Codes . . . 280
General SDK Error Codes . . . 282
P630i/P640i Specific Error Codes . . . 283
GemCore Error Codes . . . 285
MIFARE Error Codes . . . 287
Graphic Error Codes . . . 290
UHF Error Codes . . . 292

Appendix B • Data Types 295
Appendix C • Magnetic Encoders 297
Magnetic Encoders 298
Encoder Operation 298
Write 298
Read 298
Encoder Default Configuration 299
Appendix D • Bar Codes 301
Code 39 (Code 3 of 9) 302
Interleaved 2 of 5 (Code I 2/5) 303
Industrial 2 of 5 (Code 2/5) 304
EAN-8 305
EAN-13 306
UPC-A 307
Code 128, Subsets B & C 308
Appendix E • Worldwide Support 311
Appendix F • Function Index 313

1 Introduction

About This Manual

This manual contains information for software developers intending to write applications for Zebra card printers. The application programming interface (API) provides functions to access card printer features.

Important • The API depends on Zebra printer drivers being installed.

The Zebra printer drivers run on the following Windows Operating Systems:

- Windows XP Professional with Service Pack 2
- Windows 2000 with Service Pack 4
- Windows Server 2003 Service Pack 2
- Windows Vista

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 Card Printers. The following table shows the supported functions (Printer, Graphic, and Smart Cards) for the associated printer model:

MODELS	FUNCTIONS				
	Printer	Graphic	GemCore Smart Card	MIFARE Smart Card	UHF Smart Card
P100i [1]	√	√	√	√	–
P110i [1]	√	√	–	–	–
P120i [2]	√	√	–	–	–
P330i [1]	√	√	√	√	√
P430i [2]	√	√	√	√	√

1 = single-sided printing
2 = dual-sided printing
3 = dual-sided printing, single-sided laminating
4 = dual-sided printing, dual-sided laminating

Communication Ports

- USB 2.0
- Ethernet

SDK Elements

Printer

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

Graphics

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

GemCore

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

MIFARE

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

UHF

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

Installation

Directory Structure

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

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

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

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

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

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

System Directories

SDK dll files should be placed in the system directory.

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

Card-Handling

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

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

   For color, print:

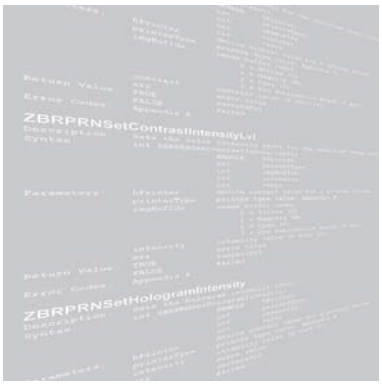
   Yellow
   Magenta
   Cyan
   Black
   Clear Varnish

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

   For color, print:

   Yellow
   Magenta
   Cyan
   Black
   Clear Varnish
   Hologram Lamination

9. Eject Card


ZBRPRNSetContrastIntensityLvl
ZBRPRNSetHologramIntensity

2 Printer Functions

Introduction

This section contains information for software developers intending to write applications for Zebra card printers. The Application Programming Interface (API) provides fuCard Printernctions to access card printer features.

Required Skills

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

Zebra Card Printers

- P110i
- P120i
- P330i
- P430i

Communication Ports

- USB 2.0
- Ethernet

Printer SDK Elements

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

Installation

Directory Structure

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

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

System Directories

SDK dll files should be placed in the system directory.

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

Function List

SDK Specific Function 12
ZBRPRNGetSDKVer 12
ZBRPRNGetSDKVsn 13
Printer Driver Handle Functions 14
ZBRGetHandle 14
ZBRCloseHandle 15
Printer Command Functions 16
ZBRPRNSendCmd 16
ZBRPRNSendCommand 17
ZBRPRNSendCmdEx 18
ZBRPRNMultipleCmd 19
ZBRPRNPrintPrnFile 20
Status Functions 21
ZBRPRNClrErrStatusLn 21
ZBRPRNGetPrintCount 22
ZBRPRNGetPrinterSerialNumber 23
ZBRPRNGetPrinterSerialNumb 24
ZBRPRNGetPrinterOptions 25
ZBRPRNGetPrintHeadSerialNumber 26
ZBRPRNGetPrintHeadSerialNumb 27
ZBRPRNGetOpParam 28
ZBRPRNGetPrinterStatus 29
ZBRPRNGetSensorStatus 30
ZBRPRNIsPrinterReady 31
Cleaning Functions 32
ZBRPRNChkDueForCleaning 32
ZBRPRNStartCleaningSeq 33
ZBRPRNStartCleaningCardSeq 34
ZBRPRNGetCleaningParam 35
ZBRPRNSetCleaningParam 36
Printer Setup Functions 37
ZBRPRNResetPrinter 37
ZBRPRNSelfAdj 38
ZBRPRNGetChecksum 39
ZBRPRNSetCardFeedingMode 40
ZBRPRNSetPrintHeadResistance 41
ZBRPRNClrMediaPath 42
ZBRPRNImmediateParamSave 43
ZBRPRNSetRelativeXOffset 44
ZBRPRNSetRelativeYOffset 45
ZBRPRNSetStartPrintXOffset 46

ZBRPRNSetStartPrintYOffset 47
ZBRPRNSetStartPrintSideBOffset 48
ZBRPRNSetStartPrintSideBXOffset 49
ZBRPRNSetStartPrintSideBYOffset 50
Image Buffer Functions 51
ZBRPRNSetColorContrast 51
ZBRPRNSetContrastIntensityLvl 52
ZBRPRNSetHologramIntensity 53
ZBRPRNSetMonoContrast 54
ZBRPRNSetMonoIntensity 55
ZBRPRNClrSpecifiedBmp 56
ZBRPRNClrMonoImgBuf 57
ZBRPRNClrMonoImgBufs 58
ZBRPRNClrColorImgBufs 59
ZBRPRNClrColorImgBuf 60
ZBRPRNPrintMonoImgBuf 61
ZBRPRNPrintMonoImgBufEx 62
ZBRPRNPrintColorImgBuf 63
ZBRPRNPrintClearVarnish 64
ZBRPRNPrintVarnish 65
ZBRPRNPrintVarnishEx 66
ZBRPRNPrintHologramOverlay 67
ZBRPRNPrintCardPanel 68
ZBRPRNPrintMonoPanel 69
ZBRPRNWriteBox 70
ZBRPRNWriteBoxEx 71
ZBRPRNWriteText 72
ZBRPRNWriteTextEx 73
ZBRPRNSetEndOfPrint 74
Position Card Functions 75
ZBRPRNMovePrintReady 75
ZBRPRNReversePrintReady 76
ZBRPRNEjectCard 77
ZBRPRNFlipCard 78
ZBRPRNMoveCard 79
ZBRPRNMoveCardBkwd 80
ZBRPRNMoveCardFwd 81
ZBRPRNResync 82
Test Card Function 83
ZBRPRNPrintTestCard 83
Barcode Card Function 84
ZBRPRNWriteBarCode 84

Magnetic Encoder Functions 85
ZBRPRNSetEncodingDir 85
ZBRPRNSetTrkDensity 86
ZBRPRNResetMagEncoder 87
ZBRPRNSetEncoderCoercivity 88
ZBRPRNSetMagEncodingStd 89
ZBRPRNReadMag 90
ZBRPRNReadMagByTrk 91
ZBRPRNWriteMag 92
ZBRPRNWriteMagByTrk 93
ZBRPRNWriteMagPassThru 94

SDK Specific Function

ZBRPRNGetSDKVer

Description: Returns the SDK dll version.

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

Parameters:

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

ZBRPRNGetSDKVsn

Description: Returns the SDK dll version.

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

Parameters:

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

Printer Driver Handle Functions

ZBRGetHandle

Description: Gets a handle for a printer driver.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRCloseHandle

Description: Closes a handle for a printer driver.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

Printer Command Functions

ZBRPRNSendCmd

Description: Sends a command to a printer.

Syntax:
```
int ZBRPRNSendCmd(
                        HANDLE        hPrinter,
                        int           printerType,
                        char          *cmd,
                        int           *err)
```

Parameters:

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

Comments: If the leading character in the command buffer is not an “escape” character, one is inserted.

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNSendCommand

Description: Sends the given command.

Syntax:
```
int ZBRPRNSendCommand(
                        HANDLE      hPrinter,
                        int         printerType,
                        char        *cmd,
                        int         *err)
```

Parameters:

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

Comments: If the leading character in the command buffer is not an “escape” character, one is inserted.

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNSendCmdEx

Description: Sends a command to a printer and returns the response.

Syntax:
```
int ZBRPRNSendCmdEx(
                        HANDLE        hPrinter,
                        int           printerType,
                        char          *cmd,
                        char          *response
                        int           *respSize
                        int           *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cmd	[in] pointer to command buffer
response	[out]pointer to response buffer
respSize	[out]pointer to number of bytes returned
err	[out]pointer to returned error code

Comments: If the leading character in the command buffer is not an “escape” character, one is inserted.

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNMultipleCmd

Description: Repeats a command a specified number of times.

Syntax:
```
int ZBRPRNMultipleCmd(
                        HANDLE      hPrinter,
                        int         printerType,
                        int         numb,
                        char        *cmd,
                        int         *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
numb	[in] number of times to send the command
cmd	[in] pointer to command buffer
err	[out]pointer to returned error code

Comments: If the leading character in the command buffer is not an “escape” character, one is inserted.

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNPrintPrnFile

Description: Prints an *.prn file.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

Status Functions

ZBRPRNClrErrStatusLn

Description: Clears the error status line.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNGetPrintCount

Description: Gets the total number of cards printed.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNGetPrinterSerialNumber

Description: Gets the printer serial number.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNGetPrinterSerialNumb

Description: Gets the printer serial number.

Syntax:
```
int ZBRPRNGetPrinterSerialNumb(
                          HANDLE        hPrinter,
                          int           printerType,
                          char          *serialNumb,
                          int           *sizeNeeded,
                          int           *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
serialNumb	[out]pointer to buffer to store the serial number of the printer
sizeNeeded	[out]pointer to number of bytes written to the serialNumb buffer
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNGetPrinterOptions

Description: Gets the printer options.

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

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
options	[out]pointer to options buffer: B = Contact smart card encoder C = Contact & HID smart card encoder D = Contact & MIFARE smart card encoder E = Contact smart card station F = HID smart card encoder H = MIFARE smart card encoder M = Magnetic encoder V = indicates firmware version
respSize	[out]pointer to response size
err	[out]pointer to returned error code

Example Response: P330iM V1.04.08<ACK>

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNGetPrintHeadSerialNumber

Description: Gets the print head serial number.

Syntax:
```
int ZBRPRNGetPrintHeadSerialNumber(
                        HANDLE        hPrinter,
                        int           printertype,
                        char          *serialNumb,
                        int           *respSize,
                        int           *err)
```

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

Return Value:		
	TRUE	successful
	FALSE	failed

Error Codes: Appendix A

ZBRPRNGetPrintHeadSerialNumb

Description: Gets the print head serial number.

Syntax:
```
int ZBRPRNGetPrintHeadSerialNumb(
                        HANDLE        hPrinter,
                        int           printerType,
                        char          *serialNumb,
                        int           *sizeNeeded,
                        int           *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
serialNumb	[out]pointer to buffer to store the print head serial number
sizeNeeded	[out]pointer to number of bytes written to the serialNumb buffer
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNGetOpParam

Description: Gets the operational parameters.

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

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

Return Value:		
	TRUE	successful
	FALSE	failed

Error Codes: Appendix A

Operational Parameters:

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

ZBRPRNGetPrinterStatus

Description: Returns the current printer error code status.

Note: This function only supports USB communication.

Syntax: int ZBRPRNGetPrinterStatus(
int *statusCode)

Parameters:

statusCode	[out]pointer to current error code status

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNGetSensorStatus

Description: Retrieves the state of various sensors for the P1xx value line printers.

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

Parameters:

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

Note: Supports P1xx value line printers only.

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNIsPrinterReady

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

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

Parameters:

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

Return Value:

TRUE	Printer is ready
FALSE	Printer is currently executing a print job

Error Codes: Appendix A

Cleaning Functions

ZBRPRNChkDueForCleaning

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

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNStartCleaningSeq

Description: Starts a cleaning sequence.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNStartCleaningCardSeq

Description: Starts card cleaning sequence.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNGetCleaningParam

Description: Gets cleaning values.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNSetCleaningParam

Description: Sets the cleaning parameters.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

Printer Setup Functions

ZBRPRNResetPrinter

Description: Resets printer.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNSelfAdj

Description: Initiates a printer Self-Adjust Sequence.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNGetChecksum

Description: Gets the firmware checksum.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNSetCardFeedingMode

Description: Sets the card feeding mode.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNSetPrintHeadResistance

Description: Sets the print head resistance.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNClrMediaPath

Description: Clears the media path.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNImmediateParamSave

Description: Immediate save of parameters to flash memory.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNSetRelativeXOffset

Description: Offsets X-axis Print Origin plus or minus dot values from current setting.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNSetRelativeYOffset

Description: Offsets Y-axis Print Origin plus or minus dot values from current setting.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNSetStartPrintXOffset

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

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

Parameters:

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

Note: 300 dots per inch

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNSetStartPrintYOffset

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

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

Parameters:

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

Note: 300 dots per inch

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNSetStartPrintSideBOffset

Description: Alters the Horizontal (X-axis) or Vertical (Y-axis) Start Print Offset Point, in dots.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

Note: This function is supported by P120i only.

ZBRPRNSetStartPrintSideBXOffset

Description: Sets the card side B X-axis start print offset point.

Note: This function only supports P120i printers.

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

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

Note: 300 dots per inch

Return Value:		
	TRUE	successful
	FALSE	failed

Error Codes: Appendix A

ZBRPRNSetStartPrintSideBYOffset

Description: Sets the card side B Y-axis start print offset point.

Note: This function only supports P120i printers.

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

Parameters:

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

Note: 300 dots per inch

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

Image Buffer Functions

ZBRPRNSetColorContrast

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

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNSetContrastIntensityLvl

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

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNSetHologramIntensity

Description: Sets the hologram intensity level.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNSetMonoContrast

Description: Sets the monochrome contrast level.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNSetMonoIntensity

Description: Sets the monochrome ribbon transfer intensity level.

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

Parameters:

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

Return Value:
TRUEsuccessful
FALSEfailed

Error Codes: Appendix A

ZBRPRNClrSpecifiedBmp

Description: Clears the specified color buffer.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNClrMonoImgBuf

Description: Clears the monochrome image buffer.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNClrMonoImgBufs

Description: Clears the monochrome image buffers.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNClrColorImgBufs

Description: Clears all of the color image buffers.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNClrColorImgBuf

Description: Clears the specified color buffer.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNPrintMonoImgBuf

Description: Prints the monochrome buffer and ejects the card.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNPrintMonoImgBufEx

Description: Prints the monochrome buffer.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNPrintColorImgBuf

Description: Print the specified color image buffer.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNPrintClearVarnish

Description: Prints Varnish from all of Image Buffers and ejects the card.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNPrintVarnish

Description: Print with clear varnish.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNPrintVarnishEx

Description: Print with clear varnish.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNPrintHologramOverlay

Description: Prints the inverse of image data and ejects the card.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNPrintCardPanel

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

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNPrintMonoPanel

Description: Prints the monochrome buffer and ejects the card.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNWriteBox

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

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

Parameters:

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

Note: 300 dots per inch

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNWriteBoxEx

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

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

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
startX	[in] start X position in dots
startY	[in] start Y position in dots
width	[in] width of the box in dots
height	[in] height of the box in dots
thickness	[in] line thickness in dots
gMode	[in] graphic mode: 0 = clear print area and load reverse bit map image 1 = clear print area and load bit map image 2 = merge bit map image with print area
isVarnish	[in] 1 = use varnish overlay
err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNWriteText

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

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

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
startX	[in] start X position in dots
startY	[in] start Y position in dots
rotation	[in] rotation: 0 = origin lower left no rotation 1 = origin lower left 90 degrees 2 = origin lower left 180 degrees 3 = origin lower left 270 degrees 4 = origin center no rotation 5 = origin center 90 degrees 6 = origin center 180 degrees 7 = origin center 270 degrees
isBold	[in] 1 = bold
height	[in] height in dots of the text box: 104 = 28 point normal 140 = 28 point bold
text	[in] pointer to text buffer
err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNWriteTextEx

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

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

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
startX	[in] start X position in dots
startY	[in] start Y position in dots
rotation	[in] rotation: 0 = origin lower left no rotation 1 = origin lower left 90 degrees 2 = origin lower left 180 degrees 3 = origin lower left 270 degrees 4 = origin center no rotation 5 = origin center 90 degrees 6 = origin center 180 degrees 7 = origin center 270 degrees
isBold	[in] 1 = bold
width	[in] width in dots of the text box, if 0 scales according to height
height	[in] height in dots of the text box: 104 = 28 point normal 140 = 28 point bold
gMode	[in] graphic mode: 0 = clear print area and load reverse bit map image 1 = clear print area and load bit map image 2 = merge bit map image with print area
text	[in] pointer to text buffer
isVarnish	[in] 1 = use varnish overlay
err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNSetEndOfPrint

Description: Specifies printing width, x axis.

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

Parameters:

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

Note: 300 dots per inch

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

Position Card Functions

ZBRPRNMovePrintReady

Description: Moves a card to the print ready position.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNReversePrintReady

Description: Moves the card back to the ready position.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNEjectCard

Description: Moves the card to the output hopper.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNFlipCard

Description: Flips a card.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNMoveCard

Description: Moves the card a specified distance.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNMoveCardBkwd

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

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

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
steps	[in] number of steps to move the card backwards
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNMoveCardFwd

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

Syntax: `int ZBRPRNGetMoveCardFwd(`

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

Parameters:		
	hPrinter	[in] printer driver handle
	printerType	[in] printer type value, see Appendix B
	steps	[in] number of steps to move the card forward
	err	[out]pointer to returned error code

Return Value:		
	TRUE	successful
	FALSE	failed

Error Codes: Appendix A

ZBRPRNResync

Description: Resynchronize the card position under the print head.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

Test Card Function

ZBRPRNPrintTestCard

Description: Prints a test card.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

Barcode Card Function

ZBRPRNWriteBarCode

Description: Writes a barcode to the monochrome buffer.

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

Parameters:

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

Note: 300 dots per inch

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

Magnetic Encoder Functions

ZBRPRNSetEncodingDir

Description: Sets the magnetic encoding direction.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNSetTrkDensity

Description: Sets track encoding density.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNResetMagEncoder

Description: Resets the magnetic encoder.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNSetEncoderCoercivity

Description: Sets the encoder coercivity.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNSetMagEncodingStd

Description: Sets encoding standard.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNReadMag

Description: Reads the specified tracks.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNReadMagByTrk

Description: Reads a specified track.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNWriteMag

Description: Encodes the specified tracks.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNWriteMagByTrk

Description: Encodes data on a specified track.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNWriteMagPassThru

Description: Supports the magnetic pass through commands; see example on the next page.

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

Parameters:

hDC	[in] handle to the printer's graphical context
printerType	[in] printer type value, see Appendix B
trkNumb	[in] track number: 1 = track 1 2 = track 2 3 = track 3
trkData	[in] pointer to data buffer
err	[out]pointer to returned error code

Note: Returns Error Code 40 (invalid magnetic data) if attempting to encode a track with no data.

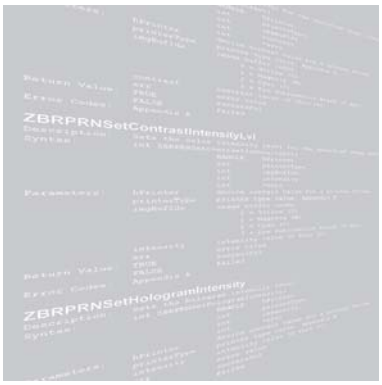
Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNWriteMagPassThru Example:

```
       // Get Printer Handle

getHandle = (funcGetHandle)GetProcAddress(dllPrnHandle, "ZBRGetHandle");
ret = getHandle(&prnHandle, "Zebra P330i USB Card Printer", &prnType, &errValue);

       // Init ZBRGraphics

initGraphics = (funcInitGraphics)GetProcAddress(dllGdiHandle, "ZBRGDIInitGraphics");
ret = initGraphics("Zebra P330i USB Card Printer", &hDC, &errValue);

       // Create Mag Track Buffers

for (int i=0; i < sizeof(trkBuf1); i++) {
       trkBuf1[i] = 0;
       trkBuf2[i] = 0;
       trkBuf3[i] = 0;
}

       // Load data to encode into the track buffers
for (i=0; i<8; i++) {
       trkBuf1[i] = 0x30 + i;
       trkBuf2[i] = 0x31 + i;
       trkBuf3[i] = 0x32 + i;
}

/* Track1     = 0x01 (001)
   Track2     = 0x02 (010)
   Track3     = 0x04 (100)
   All Tracks = 0x07 (111)
*/

       // Load data to encode

magPassThru = (funcMagPassThru)GetProcAddress(dllPrnHandle, "ZBRPRNWriteMagPassThru");
ret = magPassThru(hDC, prnType, 0x07, trkBuf1, trkBuf2, trkBuf3, &errValue);

       // Start print/encode job

printGraphics = (funcPrintGraphics)GetProcAddress(dllGdiHandle, "ZBRGDIPrintGraphics");
ret = printGraphics(hDC, &errValue);

       // Close print/encode job

closeGraphics = (funcCloseGraphics)GetProcAddress(dllGdiHandle, "ZBRGDICloseGraphics");
ret = closeGraphics(hDC, &errValue);

       // Close handle to ZBRPrinter

closeHandle = (funcCloseHandle)GetProcAddress(dllPrnHandle, "ZBRCloseHandle");
ret = closeHandle(prnHandle, &errValue);
```


ZBRPRNSetContrastIntensityLvl
ZBRPRNSetHologramIntensity

3 Graphic Functions

Introduction

This section contains information for software developers intending to write graphic applications for Zebra card printers. The application programming interface (API) provides a collection of graphic functions compatible with ID card printers.

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

- P110i
- P120i
- P330i
- P430i

Communication Ports

- USB 2.0
- Ethernet

SDK Elements

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

Installation

Directory Structure

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

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

System Directories

SDK dll files should be placed in the system directory.

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

Function List

SDK Specific Function 100
ZBRGDIGetSDKVer 100
ZBRGDIGetSDKVsn 101
Initialization Functions 102
ZBRGDIInitGraphics 102
ZBRGDIInitGraphicsEx 103
ZBRGDIInitGraphicsFromPrintDlg 104
ZBRGDICloseGraphics 105
ZBRGDIClearGraphics 106
ZBRGDIStartPage 107
ZBRGDIEndPage 108
Print Functions 109
ZBRGDIPreviewGraphics 109
ZBRGDIPrintGraphics 110
ZBRGDIPrintGraphicsEx 111
ZBRGDIPrintFilePos 112
ZBRGDIPrintFileRect 113
ZBRGDIIsPrinterReady 114
Draw Functions 115
ZBRGDIDrawText 115
ZBRGDIDrawTextUnicode 116
ZBRGDIDrawTextEx 117
ZBRGDIDrawTextRect 118
ZBRGDIDrawTextRectEx 119
ZBRGDIDrawLine 120
ZBRGDIDrawImage 121
ZBRGDIDrawImagePos 122
ZBRGDIDrawImageRect 123
ZBRGDIDrawRectangle 124
ZBRGDIDrawEllipse 125
ZBRGDIDrawBarCode 126

SDK Specific Function

ZBRGDIGetSDKVer

Description: Returns the SDK dll version.

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

Parameters:

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

ZBRGDIGetSDKVsn

Description: Returns the Graphics SDK version.

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

Parameters:

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

Error Codes: Appendix A

Initialization Functions

ZBRGDIInitGraphics

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

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGDIInitGraphicsEx

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

Syntax:

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGDIInitGraphicsFromPrintDlg

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

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGDICloseGraphics

Description: Releases device context and graphic buffer memory.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGDIClearGraphics

Description: Clears the graphic buffer.

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

Parameters:	err	[out]pointer to returned error code
Return Value:	TRUE	successful
	FALSE	failed

Error Codes: Appendix A

ZBRGDIStartPage

Description: Calls Win32 StartPage Function.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGDIEndPage

Description: Calls Win32 EndPage Function.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

Print Functions

ZBRGDIPreviewGraphics

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

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

Parameters:

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

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGDIPrintGraphics

Description: Prints the graphic buffer.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGDIPrintGraphicsEx

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

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGDIPrintFilePos

Description: Prints an image file.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGDIPrintFileRect

Description: Prints an image file within the rectangle boundaries.

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

Parameters:

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

Note: 300 dots per inch

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGDIIsPrinterReady

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

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

Parameters:

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

Return Value:

TRUE	Printer is ready
FALSE	Printer is currently executing a print job

Error Codes: Appendix A

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

Draw Functions

ZBRGDIDrawText

Description: Draws text in the graphic buffer.

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

Parameters:

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

Note: 300 dots per inch

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGDIDrawTextUnicode

Description: Draws Unicode text in the graphic buffer.

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

Parameters:

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

Note: 300 dots per inch

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGDIDrawTextEx

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

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

Parameters:

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

Note: 300 dots per inch

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGDIDrawTextRect

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

Syntax:
```
int ZBRGDIDrawTextRect(
                        int         x,
                        int         y,
                        int         width,
                        int         height,
                        int         alignment,
                        char        *text,
                        char        *font,
                        int         fontSize,
                        int         fontStyle,
                        int         color,
                        int         *err)
```

Parameters:

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

Note: 300 dots per inch

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGDIDrawTextRectEx

Description: Draws the given text in the image buffer within the bounds of the specified rectangle. The text will be automatically moved to the next line if it does not fit the given width.

Syntax:
```
int ZBRGDIDrawTextRectEx(
                        int           x,
                        int           y,
                        int           width,
                        int           height,
                        int           angle,
                        int           alignment,
                        char          *text,
                        char          *font,
                        int           fontSize,
                        int           fontStyle,
                        int           color,
                        int           *err)
```

Parameters:

x	[in] x coordinate of the rectangle bounding the text
y	[in] y coordinate of the rectangle bounding the text
width	[in] width of the rectangle bounding the text
height	[in] height of the rectangle bounding the text
angle	[in] text rotation angle
alignment	[in] alignment: 4 = center justified 5 = left justified 6 = right justified
text	[in] pointer to text that needs to be drawn
font	[in] pointer to font name
fontsize	[in] font size
fontStyle	[in] values ORed to form font style: 0x01 = bold 0x02 = italic 0x04 = underline 0x08 = strikethrough
color	[in] RGB value
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGDIDrawLine

Description: Draws a line in the graphic buffer.

Syntax:
```
int ZBRGDIDrawLine(
                        int         x1,
                        int         y1,
                        int         x2,
                        int         y2,
                        int         color,
                        float       thickness,
                        int         *err)
```

Parameters:

x1	[in] starting x position for the line in dots
y1	[in] starting y position for the line in dots
x2	[in] ending x position for the line in dots
y2	[in] ending y position for the line in dots
color	[in] RGB value
thickness	[in] thickness in dots
err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGDIDrawImage

Description: Places a file image in the graphic buffer.

Syntax:
```
int ZBRGDIDrawImage(
                        char        *filename,
                        int         x,
                        int         y,
                        int         *err)
```

Parameters:

	filename	[in] pointer to name of the file that contains the image
	x	[in] x position, top-left corner of image
	y	[in] y position, top-left corner of image
	err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGDIDrawImagePos

Description: Places a file image in the graphic buffer.

Syntax:
```
int ZBRGDIDrawImagePos(
                        char        *filename,
                        int         position,
                        int         *err)
```

Parameters:

	filename	[in] pointer to name of the file that contains the image
	position	[in] position 0 = upper left 1 = lower left 2 = upper right 3 = lower right 4 = centered
	err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGDIDrawImageRect

Description: Places a file image in the graphic buffer within rectangle boundaries.

Syntax:
```
int ZBRGDIDrawImageRect(
                        char        *filename,
                        int         x,
                        int         y,
                        int         width,
                        int         height,
                        int         *err)
```

Parameters:

filename	[in] pointer to name of the file that contains the image
x	[in] x position, top-left corner of rectangle
y	[in] y position, top-left corner of rectangle
width	[in] rectangle width in dots
height	[in] rectangle height in dots
err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGDIDrawRectangle

Description: Draws a rectangle in the graphic buffer.

Syntax:
```
int ZBRGDIDrawRectangle(
                        int         x,
                        int         y,
                        int         width,
                        int         height,
                        float       thickness,
                        int         color,
                        int         *err)
```

Parameters:

x	[in] x position, top-left corner of rectangle
y	[in] y position, top-left corner of rectangle
width	[in] rectangle width in dots
height	[in] rectangle height in dots
thickness	[in] line thickness for the rectangle
color	[in] RGB color value
err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGDIDrawEllipse

Description: Draws an ellipse in the graphic buffer.

Syntax:
```
int ZBRGDIDrawEllipse(
                        int         x,
                        int         y,
                        int         width,
                        int         height,
                        float       thickness,
                        int         color,
                        int         *err)
```

Parameters:

x	[in] x position, top-left corner of rectangle
y	[in] y position, top-left corner of rectangle
width	[in] width of the ellipse
heigth	[in] height of the ellipse
thickness	[in] line thickness for the ellipse in dots
color	[in] RGB color value
err	[out]pointer to returned error code

Note: 300 dots per inch

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGDIDrawBarCode

Description: Writes a barcode to the monochrome buffer.

Syntax:
```
int ZBRGDIDrawBarCode(
                        int           X,
                        int           Y,
                        int           rotation,
                        int           barcodeType,
                        int           barWidthRatio,
                        int           barcodeMultiplier,
                        int           barcodeHeight,
                        int           textUnder,
                        char          *barcodeData,
                        int           *err)
```

Parameters:

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

Note: 300 dots per inch

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

4 GemCore Functions

Introduction

This section contains information for software developers intending to write applications for Synchronous and ISO 7816-3 compliant contact smart cards using Zebra card printer's internal smart card readers. The application programming interface (API) provides functions to access the internal smart card features.

Required Skills

- Experience in developing applications for the Microsoft Windows environment
- Experience in developing applications using dynamic link libraries (dll)
- Experience with ISO 7816-3 compliant smart cards

Zebra Card Printers

- P330i
- P430i

Communication Ports

- USB 2.0
- Ethernet

SDK Elements

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

Installation

Directory Structure

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

doc directory contains SDK documentation

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

sample directory contains example applications

System Directories

SDK dll files should be placed in the system directory.

Example -- XP

(Disk Drive):\WINDOWS\system32\

Function List

SDK Specific Function 130
ZBRGCGetSDKVer 130
ZBRGCGetSDKVsn 131
Printer Functions 132
ZBRGetHandle 132
ZBRCloseHandle 133
ZBRGCStartCard 134
ZBRGCEndCard 135
ZBRGCEndCardEx 136
Card Specific Functions 137
ZBRGCCardPowerUp 137
ZBRGCCardPowerUpEx 138
ZBRGCCardPowerUpPPS 139
ZBRGCCardPowerDown 140
ZBRGCReaderPowerDown 141
ZBRGCExchangeData 142
ZBRGCExchangeAPDU 143
ZBRGCISOInput 144
ZBRGCISOOutput 145
ZBRGCCardStatus 146
Reader Specific Functions 147
ZBRGCSetCardType 147
ZBRGCDirectory 148
ZBRGCReadFirmwareVer 149
ZBRGCReadFirmwareVsn 150
ZBRGCGetOpMode 151
ZBRGCSetOpMode 152
ZBRGCGetTimeout 153
ZBRGCSetTimeout 154

SDK Specific Function

ZBRGCGetSDKVer

Description: Returns the SDK version numbers.

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

Parameters:

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

ZBRGCGetSDKVsn

Description: Return SDK version number.

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

Parameters:

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

Error Codes: Appendix A

Printer Functions

ZBRGetHandle

Description: Gets a handle for a printer driver.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRCloseHandle

Description: Closes a handle to a printer driver.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGCStartCard

Description: Positions a card for internal contact smart card encoding.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGCEndCard

Description: Indicate that encoding is done.

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

Parameters:

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

Note: It is important to call this function after all other communication with the smart card reader is finished.

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGCEndCardEx

Description: Indicate that encoding is done if eject is true the card is ejected.

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

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
eject	[in] 1 = eject card after encoding
err	[out]pointer to returned error code

Note: It is important to call this function after all other communication with the smart card reader is finished.

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

Card Specific Functions

ZBRGCCardPowerUp

Description: Powers up and resets an ISO 7816-3 Microprocessor card.

Syntax:
```
int ZBRGCCardPowerUp(
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            *atr,
                        int             *atrSize,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
atr	[out]pointer to response buffer
atrSize	[out]pointer to byte count of the response
err	[out]pointer to returned error code

Note: Returns the ATR (Answer To Reset) buffer.

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGCCardPowerUpEx

Description: Powers up and resets an ISO 7816-3 Microprocessor card.

Syntax:
```
int ZBRGCCardPowerUpEx(
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            cfg,
                        byte            *atr,
                        int             *atrSize,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cfg	[in] card configuration byte; see *Details:* X0XXX001 X0XXX010 X0XXX100 X0XXX011 X0XXX110 X0XXX111 0000XXXX 0001XXXX 0010XXXX 1111XXXX 00001000 X0XX1XXX 11111XXX
atr	[out]pointer to response buffer
atrSize	[out]pointer to byte count of the response
err	[out]pointer to returned error code

Note: Returns the ATR (Answer To Reset) buffer.

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

Card Configuration Details:

X0XXX001 –Class A: Vcc for Card is 5V
X0XXX010 –Class B: Vcc for Card is 3V
X0XXX100 –Class C: Vcc for Card is 1.8V
X0XXX011 –Class AB: Vcc for Card is 5V or 3V
X0XXX110 -Class BC: Vcc for card is 3V or 1.8V
X0XXX111 -Class ABC: Vcc for Card is 5V, 3V, or 1.8V
0000XXXX –Operation is compatible with OROS2.2X
0001XXXX –Reset and no PPS management. The reader stays at 9600 bps if the card is in negotiable mode.
0010XXXX –Reset and automatic PPS management.The reader uses the highest speed proposed by the card. Change to T=1 proocol if there is a choice between T=0 and T=1.
1111XXXX –Manual PPS management. This command does not reset the card. It must be preceded by a Power Up command with the CFG parameter set to 0001XXXX. The parameters from PPS0 to PCK are sent to the card at 9600 bps. If PCK is omitted, it is computed and added by the reader. If the card responds with PPS Response the reader is configured using the parameters returned.
00001000 –Valid only if T=1 is the current protocol; otherwise, no action occurs. An S-IFS block exchange is initiated by the reader. The IFSD (max length of INF field accepted by the reader sent to the card is the value of parameter PPS0. No other parameters are allowed.
X0XX1XXX
11111XXX –If the selected protocol after the ATR or the PPS exchange is T=1, the reader initiates an S-IFS block exchange. The IFSD value indicated to the card is FEh. After a command reset with no PPS and with IFSD exchange, a command of manual PPS management is invlaid.

ZBRGCCardPowerUpPPS

Description: Powers up and resets an ISO 7816-3 microprocessor card utilizing manual PPS management.

Syntax:
```
int ZBRGCCardPowerUpPPS(
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            *ppsParams,
                        int             ppsSize,
                        byte            *atr,
                        int             *atrSize,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
ppsParams	[in] pointer to PPS parameters [PPS0, PPS1, PPS2, PPS3][PCK] - refer to ISO 7816-3
ppsSize	[in] size of the PPS parameter buffer
atr	[out] pointer to ATR response buffer
atrSize	[out] pointer to byte count of ATR response
err	[out] pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

Example: Passing the pps parameters into the function.

```
BYTE atr[50], ppsParams[2];
memset(atr, 0, 50);
memset(pps, 0, 2);

pps[0] = 0x10; // PPS0
pps[1] = 0x18; // PPS1 for 1152Kbps@3.58MHz
int atrSize = 0, ppsSize = 2, err = 0, retVal = 0;

retVal = ZBRGCCardPowerUpPPS(hPrinter, printerType, ppsParams, ppsSize, atr,
                             &atrSize, &err);
```

ZBRGCCardPowerDown

Description: Powers down an ISO 7816-3 Microprocessor card.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGCReaderPowerDown

Description: This function powers down the reader.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGCExchangeData

Description: Sends data to the reader and receives a response.

Syntax:
```
int ZBRGCExchangeData(
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            *dataIn,
                        int             dataInSize,
                        byte            *dataOut,
                        int             dataOutSize,
                        int             *respSize,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
dataIn	[in] pointer to APDU buffer
dataInSize	[in] size of the APDU buffer
dataOut	[out]pointer to response buffer
dataOutSize	[in] size of response buffer
respSize	[out]pointer to byte count of the response
err	[out]pointer to returned error code

Note: The data has to be in accordance to the commands specified by the GemCore™ Serial Lite PRO reference manual.

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGCExchangeAPDU

Description: Exchanges an APDU packet with an ISO 7816-3 compliant microprocessor card.

Syntax:
```
int ZBRGCExchangeAPDU(
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            *dataIn,
                        int             dataInSize,
                        byte            *dataOut,
                        int             *respSize,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
dataIn	[in] pointer to APDU buffer
dataInSize	[in] size of the APDU buffer
dataOut	[out]pointer to response buffer
respSize	[out]pointer to byte count of the response
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGCISOInput

Description: Sends ISO input commands to the card; commands that send data to an asynchronous card.

Syntax:
```
int ZBRGCISOInput(
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            *dataIn,
                        int             dataInSize,
                        byte            *dataOut,
                        int             *respSize,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
dataIn	[in] pointer to TPDU buffer
dataInSize	[in] size of the TPDU buffer
dataOut	[out]pointer to response buffer
respSize	[out]pointer to byte count of the response
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGCISOOutput

Description: Sends ISO output commands to the card; commands that retrieve data from an asynchronous card.

Syntax:
```
int ZBRGCISOInput(
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            *dataIn,
                        int             dataInSize,
                        byte            *dataOut,
                        int             *respSize,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
dataIn	[in] pointer to TPDU buffer
dataInSize	[in] size of the TPDU buffer
dataOut	[out]pointer to response buffer
respSize	[out]pointer to byte count of the response
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGCCardStatus

Description: Obtain status of the card interface. Information returned indicates:
- Type of card currently used
- Card presence • Power supply value
- Card power status
- Communication protocol (T=0 or T=1)
- Speed parameters between card and reader

Syntax:
```
int ZBRGCCardStatus(
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            *statusData,
                        int             *respSize,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
statusData	[out]pointer to buffer where status data is copied
respSize	[out]pointer to byte count of the response
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

Response Format: STAT TYPE CNF1 CNF2 CNF3 CNF4

Asynchronous Card:

STAT:
- 0000X000 – Card not inserted
- 0000X100 – Card inserted but not powered
- 0000X101 – Card inserted, power = 1.8V
- 0000X110 – Card inserted, power = 5V
- 0000X111 – Card inserted, power = 3V
- 00000XXX – T=0 protocol
- 00001XXX – T=1 protocol

TYPE: Activated card type

CNF1: TA1 (FI/DI) - T=0/T=1 Card as per ISO 7816-3

CNF2: TC1 (EGT) - T=0/T=1 Card as per ISO 7816-3

CNF3:
- WI - T=0 Card as per ISO 7816-3
- IFSC – T=1 Card as per ISO 7816-3

CNF4:
- 0x00 - T=0 Card as per ISO 7816-3
- TB3 (BWI/CWI) – T=1 Card as per ISO 7816-3

Synchronous Card:

STAT:
- 0000X000 – Card not inserted
- 0000X100 – Card inserted but not powered
- 0000X101 – Card inserted, power = 1.8V
- 0000X110 – Card inserted, power = 5V
- 0000X111 – Card inserted, power = 3V

TYPE: Activated card type

CNF1: 0x00 (RFU)

CNF2: 0x00 (RFU)

CNF3: 0x00 (RFU)

CNF4: 0x00 (RFU)

Reader Specific Functions

ZBRGCSetCardType

Description: Sets the smart card type in the reader.

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

Parameters:

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

Note: The reader itself does not have smart card detection built-in; therefore, this function must be called before card-specific functions are called. When the reader is powered up or reset, the card type defaults to standard microprocessor card (ZBR_STANDARD_78163).

Results:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGCDirectory

Description: Returns the types of cards that are handled by the reader, as well as their release numbers and characteristics of each card driver.

Syntax:
```
int ZBRGCDirectory(
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            *dirData,
                        int             *respSize,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
dirData	[out]pointer to response buffer
respSize	[out]pointer to byte count of the response
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGCReadFirmwareVer

Description: Returns firmware version of the reader.

Syntax:
```
int ZBRGCReadFirmwareVer(
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            *readerVer,
                        int             *respSize,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
readerVer	[out]pointer to response buffer
respSize	[out]pointer to byte count of the response
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGCReadFirmwareVsn

Description: Returns firmware version of the reader.

Syntax:
```
int ZBRGCReadFirmwareVsn(
                          HANDLE          hPrinter,
                          int             printerType,
                          byte            *readerVer,
                          int             *respSize,
                          int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
readerVer	[out]pointer to response buffer
respSize	[out]pointer to byte count of the response
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGCGetOpMode

Description: Returns the operating mode of the reader.

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

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
mode	[out]pointer to operating mode: 0 = ISO mode 1 = EMV mode
err	[out]pointer to returned error code

Note: The reader can operate in two modes – ISO (ZBR_ISO_MODE) or EMV (ZBR_EMV_MODE). The default mode is ISO mode.

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGCSetOpMode

Description: Sets the operating mode of the reader.

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

Parameters:

hPrinter	[in]	printer driver handle
printerType	[in]	printer type value, see Appendix B
mode	[in]	operating mode: 0 = ISO mode 1 = EMV mode
err	[out]	pointer to returned error code

Note: The reader can operate in two modes – ISO (ZBR_ISO_MODE) or EMV (ZBR_EMV_MODE). The default mode is ISO mode.

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGCGetTimeout

Description: Gets the timeout value of the reader.

Syntax:
```
int ZBRGCGetTimeOut(
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            *timeoutValue,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
timeoutValue	[out]pointer to current timeout value of the reader in seconds; 0 = infinite
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGCSetTimeout

Description: Sets the timeout value of the reader.

Syntax:
```
int ZBRGCSetTimeOut(
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            timeoutValue,
                        int             *err)
```

Parameters:

	hPrinter	[in] printer driver handle
	printerType	[in] printer type value, see Appendix B
	timeoutValue	[out]timeout value of the reader to be set to, 0 = infinite
	err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

5 MIFARE Functions

Introduction

This section contains information for software developers intending to write applications for ISO 14443-compliant contactless smart cards using Zebra card printer's internal smart card readers.

The Application Programming Interface (API) provides functions to access the internal smart card features.

Required Skills

- Experience in developing applications for the Microsoft Windows environment
- Experience in developing applications using dynamic link libraries (dll)
- Experience with Microsoft's Windows Graphics Device Interface (GDI)
- Experience with ISO 14443-compliant smart cards

Zebra Card Printers

- P330i
- P430i

Communication Ports

- USB 2.0
- Ethernet

MIFARE SDK Elements

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

Installation

Directory Structure

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

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

System Directories

SDK dll files should be placed in the system directory.

Example -- XP

(Disk Drive):\WINDOWS\system32\

Function List

DLL Function 159
ZBRGPMFGetSDKVer 159
ZBRGPMFSDKGetVer 160
Printer Functions 161
ZBRGetHandle 161
ZBRCloseHandle 162
ZBRGPMFStartCard 163
ZBRGPMFEndCard 164
ZBRGPMFEndCardEx 165
Card Functions 166
ZBRGPMF_LoadKey 166
ZBRGPMF_Authenticate 167
ZBRGPMF_Read 168
ZBRGPMF_Write 169
ZBRGPMF_SubtractValue 170
ZBRGPMF_AddValue 171
ZBRGPMF_Restore 172
ZBRGPMF_Transfer 173
Purse Card Functions 174
ZBRGPMF_B_CreatePurse 174
ZBRGPMF_B_ReadPurse 175
ZBRGPMF_B_DebitPurse 176
ZBRGPMF_B_CreditPurse 177
MAD Card Function 178
ZBRGPMF_MAD_ReadDataSector 178
Combined Card Functions 179
ZBRGPMF_C_Read 179
ZBRGPMF_C_Write 180
ZBRGPMF_C_CreateValueBlock 181
ZBRGPMF_C_ReadValue 182
ZBRGPMF_C_SubtractValue 183
ZBRGPMF_C_AddValue 184
ZBRGPMF_C_CopyValue 185
ZBRGPMF_C_SetAccessConditions 186
Reader Functions 187
ZBRGPMF_Reader_GetFirmware 187
ZBRGPMF_Reader_GetID 188
ZBRGPMF_Reader_GetModeAndGBPAddress 189
ZBRGPMF_Reader_SetMode 190
ZBRGPMF_Reader_ReadEEPROM 191
ZBRGPMF_Reader_WriteEEPROM 192
ZBRGPMF_Reader_GetParameters 193
ZBRGPMF_Reader_ActivateBuzzer 194

ZBRGPMF_Reader_ChangeMode 195
ZBRGPMF_Reader_ControlLeds 196
ZBRGPMF_Reader_OpenCaseDetection 197
ZBRGPMF_Reader_SetDelay 198
RF Functions 199
ZBRGPMF_RF_Control 199
ZBRGPMF_RF_ChangeModulationType 200
ZBRGPMF_RF_ReadModulationType 201
14443A Functions 202
ZBRGPMF_ISO14443_3_A_RequestA 202
ZBRGPMF_ISO14443_3_A_Anticollision 203
ZBRGPMF_ISO14443_3_A_Select 204
ZBRGPMF_ISO14443_3_A_Halt 205
Combined 14443A Functions 206
ZBRGPMF_ISO14443_3_A_GetCard 206
ZBRGPMF_ISO14443_3_A_RequestAllSelectA 207
ZBRGPMF_ISO14443_3_A_GetCardA_T_CL 208
ZBRGPMF_ISO14443_3_A_RequestAllSelectA_T_CL 209
14443_4_A Functions 210
ZBRGPMF_ISO14443_4_A_RequestForAnswerToSelect 210
ZBRGPMF_ISO14443_4_A_ProtocolParameterSelection 211
14443_4_A_B Functions 212
ZBRGPMF_ISO14443_4_A_B_Exchange_T_CL 212
ZBRGPMF_ISO14443_4_A_B_Deselect 213
ZBRGPMF_ISO14443_4_A_B_Poll_T_CL_Card_Removed 214
ZBRGPMF_ISO14443_4_A_B_Mode15_GetStatus 215
ZBRGPMF_GEMCORECARDI_Card_Exchange 216
ZBRGPMF_GEMCORECARDI_Exchange_IFSD 217
ZBRGPMF_GEMCORECARDI_Exchange_PPS 218
ZBRGPMF_GEMCORECARDI_PowerDown 219
ZBRGPMF_GEMCORECARDI_PowerUp 220
ZBRGPMF_GEMCORECARDI_SetCurrentSAM 221
ZBRGPMF_GEMCORECARDI_Status 222
14443B Functions 223
ZBRGPMF_ISO14443_3_B_RequestB 223
ZBRGPMF_ISO14443_3_B_SlotMarker 224
ZBRGPMF_ISO14443_3_B_Attribute 225
ZBRGPMF_ISO14443_3_B_Halt 226
Combined 14443B Function 227
ZBRGPMF_ISO14443_3_B_GetCard 227
Transparent Functions 228
ZBRGPMF_TransparentExchange 228
ZBRGPMF_TransparentExchangeTimeout 229

DLL Function

ZBRGPMFGetSDKVer

Description: Returns the SDK version numbers.

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

Parameters:

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

ZBRGPMFSDKGetVer

Description: Returns SDK version number.

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

Parameters:

major	[out]major version number of the SDK library
minor	[out]minor version number of the SDK library
engLevel	[out]the engineering level of the SDK library

Return Value: None

Printer Functions

ZBRGetHandle

Description: Gets a handle for a printer driver.

Syntax:
```
int ZBRGetHandle(
                    LPHANDLE          *hPrinter,
                    LPSTR             printerName,
                    int               *printerType,
                    int               *err)
```

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRCloseHandle

Description: Closes a printer driver handle.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMFStartCard

Description: Puts the card under reader antenna.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed, check error codes

Comment: Call this function before sending commands to the reader.

Error Codes: Appendix A

ZBRGPMFEndCard

Description: Indicates that encoding is done and ejects the card.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed, check error codes

Comment: Call this function after communication with the reader is finished and before calling ZBRCloseHandle.

Error Codes: Appendix A

ZBRGPMFEndCardEx

Description: Indicates that encoding is done and either ejects the card or moves the card to the printing location.

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

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
eject	[in] eject: 1 = eject the card after encoding 0 = position the card for printing
err	[out]pointer to returned error code

Return Value:

TRUE	successfully
FALSE	failed, check error codes

Comment: Call this function after communication with the reader is finished and before calling ZBRCloseHandle.

Error Codes: Appendix A

Card Functions

ZBRGPMF_LoadKey

Description: Load a MIFARE Key into the reader. (no card access)

Syntax:
```
int ZBRGPMF_LoadKey (
                    HANDLE          hPrinter,
                    int             printerType,
                    byte            blockNumber,
                    byte            keyAB,
                    byte            *key,
                    int             *err)
```

Parameter:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
blockNumber	[in] virtual block number = sector number X 4 0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56 or 60.
keyAB	[in] defines if key to load is Key A or Key B: 0 = KeyA 1 = KeyB
key	[in] pointer to the key to load (6 bytes).
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_Authenticate

Description: MIFARE basic card command. Performs a block authentication.

Syntax:
```
int ZBRGPMF_Authenticate (
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            cardType,
                        byte            blockNumber,
                        byte            keyAB,
                        int             *err)
```

Parameters:

	hPrinter	[in] printer driver handle
	printerType	[in] printer type value, see Appendix B
	cardType	[in] type of the current card: 0x00 = GEMEASY_8000 --> MIFARE 1K 0x02 = GEMCOMBI --> MIFARE 4K with automatic block value (Dual Interface Card) 0x03 = GEMEASY_32000 --> MIFARE 4K
	blockNumber	[in] block number to authenticate: 0 to 63 with GEMEASY_8000 0 to 255 with GEMCOMBI or GEMEASY_32000
	keyAB	[in] defines if key to load is Key A or Key B: 0 = KeyA 1 = KeyB
	err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_Read

Description: MIFARE basic card command. Read a block (16 bytes).

Syntax:
```
int ZBRGPMF_Read (
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                cardType,
                        byte                blockNumber,
                        byte                *dataBlock,
                        int                 *dataBlockSize,
                        int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardType	[in] type of the current card: 0x00 = GEMEASY_8000 --> MIFARE 1K 0x02 = GEMCOMBI --> MIFARE 4K with automatic block value (Dual Interface Card) 0x03 = GEMEASY_32000 --> MIFARE 4K
blockNumber	[in] block number to authenticate: 0 to 63 with GEMEASY_8000 0 to 255 with GEMCOMBI or GEMEASY_32000
dataBlock	[out]pointer to the data block read
dataBlockSize	[in, out] [in] pointer to number of bytes to read (16) [out]pointer to number of bytes read
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_Write

Description: MIFARE basic card command. Write a block (16 bytes).

Syntax:
```
int ZBRGPMF_Write (
                    HANDLE          hPrinter,
                    int             printerType,
                    byte            cardType,
                    byte            blockNumber,
                    byte            *dataBlock,
                    int             *dataBlockSize,
                    int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardType	[in] type of the current card: 0x00 = GEMEASY_8000 --> MIFARE 1K 0x02 = GEMCOMBI --> MIFARE 4K with automatic block value (Dual Interface Card) 0x03 = GEMEASY_32000 --> MIFARE 4K
blockNumber	[in] block number to authenticate: 0 to 63 with GEMEASY_8000 0 to 255 with GEMCOMBI or GEMEASY_32000
dataBlock	[in] pointer to the write buffer
dataBlockSize	[in, out] [in] pointer to number of bytes to write (16) [out]pointer to number of bytes written
err	[out]pointer to returned error code

Return Value:

TRUE	successfully
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_SubtractValue

Description: MIFARE basic card command. Subtract a value from a formatted value block. The result is stored in a temporary card register. Use ZBRGPMF_Transfer after this command to store the result in a block.

Syntax:
```
int ZBRGPMF_SubtractValue (
                    HANDLE          hPrinter,
                    int             printerType,
                    byte            cardType,
                    byte            blockNumber,
                    long            value,
                    int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardType	[in] type of the current card: 0x00 = GEMEASY_8000 --> MIFARE 1K 0x02 = GEMCOMBI --> MIFARE 4K with automatic block value (Dual Interface Card) 0x03 = GEMEASY_32000 --> MIFARE 4K
blockNumber	[in] block number to authenticate: 0 to 62 with GEMEASY_8000 0 to 254 with GEMCOMBI or GEMEASY_32000
value	[in] value to be subtracted (-2147483647 to +2147483648)
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_AddValue

Description: MIFARE basic card command. Add a value to a formatted value block. The result is stored in a temporary card register. Use ZBRGPMF_Transfer after this command to store the result in a block.

Syntax:
```
int ZBRGPMF_AddValue (
                    HANDLE          hPrinter,
                    int             printerType,
                    byte            cardType,
                    byte            blockNumber,
                    long            value,
                    int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardType	[in] type of the current card: 0x00 = GEMEASY_8000 --> MIFARE 1K 0x02 = GEMCOMBI --> MIFARE 4K with automatic block value (Dual Interface Card) 0x03 = GEMEASY_32000 --> MIFARE 4K
blockNumber	[in] block number to authenticate: 0 to 62 with GEMEASY_8000 0 to 254 with GEMCOMBI or GEMEASY_32000
value	[in] value to be subtracted (-2147483647 to +2147483648)
err	[out]pointer to returned error code

Note: The block cannot be 60 to 7F for GEMCOMBI (MIFARE 4K w/automatic block value).

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_Restore

Description: MIFARE basic card command. Store the value of a formatted value block in the temporary card register. Use ZBRGPMF_Transfer after this command to store the value in another block.

Syntax:
```
int ZBRGPMF_Restore (
                    HANDLE          hPrinter,
                    int             printerType,
                    byte            cardType,
                    byte            blockNumber,
                    int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardType	[in] type of the current card: 0x00 = GEMEASY_8000 --> MIFARE 1K 0x02 = GEMCOMBI --> MIFARE 4K with automatic block value (Dual Interface Card) 0x03 = GEMEASY_32000 --> MIFARE 4K
blockNumber	[in] block number to authenticate: 0 to 62 with GEMEASY_8000 0 to 254 with GEMCOMBI or GEMEASY_32000
err	[out]pointer to returned error code

Note: The block cannot be 60 to 7F for GEMCOMBI (MIFARE 4K w/automatic block value).

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_Transfer

Description: MIFARE basic card command. Transfer the contents of the temporary card register into a block.

Syntax:
```
int ZBRGPMF_Transfer (
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            cardType,
                        byte            blockNumber,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardType	[in] type of the current card: 0x00 = GEMEASY_8000 --> MIFARE 1K 0x02 = GEMCOMBI --> MIFARE 4K with automatic block value (Dual Interface Card) 0x03 = GEMEASY_32000 --> MIFARE 4K
blockNumber	[in] block number to authenticate: 0 to 62 with GEMEASY_8000 0 to 254 with GEMCOMBI or GEMEASY_32000
err	[out]pointer to returned error code

Note: The destination block must be in the same sector than the block of the previous command. The block cannot be 60 to 7F for GEMCOMBI (MIFARE 4K w/auto block value).

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

Purse Card Functions

ZBRGPMF_B_CreatePurse

Description: MIFARE Purse card command. Create a formatted purse sector. The purse is created in the 2 first block of a four blocks sector. An automatic authentication can be performed before operations.

Syntax:
```
int ZBRGPMF_B_CreatePurse (
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            cardType,
                        byte            blockNumber,
                        byte            authentication,
                        long            value,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardType	[in] type of the current card: 0x00 = GEMEASY_8000 --> MIFARE 1K 0x02 = GEMCOMBI --> MIFARE 4K with automatic block value (Dual Interface Card) 0x03 = GEMEASY_32000 --> MIFARE 4K
blockNumber	[in] block number to authenticate: 0 to 60 with GEMEASY_8000 0 to 124 with GEMCOMBI or GEMEASY_32000
authentication	[in] automatic authentication control: 0 = No Authentication 1 = Authentication KeyA 2 = Authentication KeyB
value	[in] initial purse value to format the block (-2147483647 to + 2147483648)
err	[out]pointer to returned error code

Note: blockNumber must be the first block of a 4-block sector.

The block cannot be 60 to 7F for GEMCOMBI (MIFARE 4K w/auto block value).

Return Value:

TRUE	successful
FALSE	failed, check error codes

Comment: This command is a combination of several single commands:
- Authenticate
- Write
- Read

Error Codes: Appendix A

ZBRGPMF_B_ReadPurse

Description: MIFARE Purse card command. Read the purse value content. An automatic authentication can be performed before the operation.

Syntax:
```
int ZBRGPMF_B_ReadPurse (
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            cardType,
                        byte            blockNumber,
                        byte            authentication,
                        long            *value,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardType	[in] type of the current card: 0x00 = GEMEASY_8000 --> MIFARE 1K 0x02 = GEMCOMBI --> MIFARE 4K with automatic block value (Dual Interface Card) 0x03 = GEMEASY_32000 --> MIFARE 4K
blockNumber	[in] block number to authenticate: 0 to 60 with GEMEASY_8000 0 to 124 with GEMCOMBI or GEMEASY_32000
authentication	[in] automatic authentication control: 0 = No Authentication 1 = Authentication KeyA 2 = Authentication KeyB
value	[out]pointer to purse value read (-2147483647 to + 2147483648)
err	[out]pointer to returned error code

Note: blockNumber must be the first block of a 4-block sector.

The block cannot be 60 to 7F for GEMCOMBI (MIFARE 4K w/auto block value).

Return Value:

TRUE	successful
FALSE	failed, check error codes

Comment: This command is a combination of several single commands:
- Authenticate
- Read
- Restore
- Transfer

Error Codes: Appendix A

ZBRGPMF_B_DebitPurse

Description: MIFARE Purse card command. Perform a debit operation to purse value content. An automatic authentication can be performed before the operation.

Syntax:
```
int ZBRGPMF_B_DebitPurse (
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                cardType,
                        byte                blockNumber,
                        byte                authentication,
                        long                value,
                        int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardType	[in] type of the current card: 0x00 = GEMEASY_8000 --> MIFARE 1K 0x02 = GEMCOMBI --> MIFARE 4K with automatic block value (Dual Interface Card) 0x03 = GEMEASY_32000 --> MIFARE 4K
blockNumber	[in] block number to authenticate: 0 to 60 with GEMEASY_8000 0 to 124 with GEMCOMBI or GEMEASY_32000
authentication	[in] automatic authentication control: 0 = No Authentication 1 = Authentication KeyA 2 = Authentication KeyB
value	[in] value to debit from the purse (-2147483647 to + 2147483648)
err	[out]pointer to returned error code

Note: blockNumber must be the first block of a 4-block sector.

The block cannot be 60 to 7F for GEMCOMBI (MIFARE 4K w/auto block value).

Return Value:

TRUE	successful
FALSE	failed, check error codes

Comment: This command is a combination of several single commands:

- Authenticate
- Read
- Decrement
- Restore
- Transfer

Error Codes: Appendix A

ZBRGPMF_B_CreditPurse

Description: MIFARE Purse card command. Perform a credit operation to purse value content. An automatic authentication can be performed before the operation.

Syntax:
```
int ZBRGPMF_B_CreditPurse (
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                cardType,
                        byte                blockNumber,
                        byte                authentication,
                        long                value,
                        int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardType	[in] type of the current card: 0x00 = GEMEASY_8000 --> MIFARE 1K 0x02 = GEMCOMBI --> MIFARE 4K with automatic block value (Dual Interface Card) 0x03 = GEMEASY_32000 --> MIFARE 4K
blockNumber	[in] block number to authenticate: 0 to 60 with GEMEASY_8000 0 to 124 with GEMCOMBI or GEMEASY_32000
authentication	[in] automatic authentication control: 0 = No Authentication 1 = Authentication KeyA 2 = Authentication KeyB
value	[in] value to add to the purse (-2147483647 to + 2147483648)
err	[out]pointer to returned error code

Note: blockNumber must be the first block of a 4-block sector.

The block cannot be 60 to 7F for GEMCOMBI (MIFARE 4K w/auto block value).

Return Value:

TRUE	successful
FALSE	failed, check error codes

Comment: This command is a combination of several single commands:

- Authenticate
- Read
- Increment
- Restore
- Transfer

Error Codes: Appendix A

MAD Card Function

ZBRGPMF_MAD_ReadDataSector

Description: MIFARE MAD command. Read the data from a sector of a MAD formatted card. All the operations to access data in the card are automatically performed by this command. Use this command in a loop to read all data sectors of a card. Only the data sector corresponding to the AID configured in the reader are read.

Syntax:
```
int ZBRGPMF_MAD_ReadDataSector (
                    HANDLE              hPrinter,
                    int                 printerType,
                    int                 tryTime,
                    byte                *data,
                    int                 *dataSize,
                    int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
tryTime	[in] try time duration for reader to get a MAD card 0 to 256 by step of 100ms 0 = one try only
data	[out]pointer to the read buffer
dataSize	[in, out] [in] pointer to size of data buffer (min 247) [out]pointer to number of bytes read
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Comment: The exchange timeout will be adjusted to comply with the try time duration. The reader must be configured to MAD operating mode before using this command.

Error Codes: Appendix A

Combined Card Functions

ZBRGPMF_C_Read

Description: MIFARE Combined card command. Read data from one or several blocks of a same sector. An automatic authentication can be performed or not before operation.

Syntax:
```
int ZBRGPMF_C_Read (
                    HANDLE          hPrinter,
                    int             printerType,
                    byte            cardType,
                    byte            blockNumber,
                    byte            authentication,
                    byte            *data,
                    int             *dataSize,
                    int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardType	[in] type of the current card 0x00 = GEMEASY_8000 --> MIFARE 1K 0x02 = GEMCOMBI --> MIFARE 4K with automatic block value (Dual Interface Card) 0x03 = GEMEASY_32000 --> MIFARE 4K
blockNumber	[in] block number to read from: 0 to 62 with GEMEASY_8000 0 to 254 with GEMCOMBI or GEMEASY_32000
authentication	[in] automatic authentication control: 0 = No Authentication 1 = Authentication KeyA 2 = Authentication KeyB
data	[out]pointer to the read buffer
dataSize	[in, out] [in] pointer to number of bytes to read (max 255) [out]pointer to number of bytes read
err	[out]pointer to returned error code

Note: All the data byte to be read must be in the same sector. Partial blocks can be read.

Up to 128 bytes can be read in a single operation.

Valid sectors and data lengths follow:

- MIFARE 1K
  Sector Range: 0 - 15
  Maximum data lengths:
  Sector 0: 32 bytes
  Sectors 1 - 15: 48 bytes
- MIFARE 4K
  Sector Range: 0 - 39
  Maximum data lengths:
  Sector 0: 32 bytes
  Sectors 1 - 31: 48 bytes
  Sectors 32 - 39: 240 bytes

Return Value:

TRUE	successful
FALSE	failed, check error codes

Comment: This command is a combination of several single commands:

- Authenticate
- Read

Error Codes: Appendix A

ZBRGPMF_C_Write

Description: MIFARE Combined card command. Write data into one or several blocks of a same sector. An automatic authentication can be performed or not before operation. An automatic write verification can be performed or not after operation.

Syntax:
```
int ZBRGPMF_C_Write (
                    HANDLE          hPrinter,
                    int             printerType,
                    byte            cardType,
                    byte            blockNumber,
                    byte            authentication,
                    byte            writeVerify,
                    byte            *data,
                    int             *dataSize,
                    int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardType	[in] type of the current card : 0x00 = GEMEASY_8000 --> MIFARE 1K 0x02 = GEMCOMBI --> MIFARE 4K with automatic block value (Dual Interface Card) 0x03 = GEMEASY_32000 --> MIFARE 4K
blockNumber	[in] block number to authenticate: 0 to 63 with GEMEASY_8000 0 to 255 with GEMCOMBI or GEMEASY_32000
authentication	[in] automatic authentication control: 0 = No Authentication 1 = Authentication KeyA 2 = Authentication KeyB
writeVerify	[in] perform write verification: 0 = No Write Verification 1 = Perform Write Verification
data	[in] pointer to the write buffer
dataSize	[in, out] [in] pointer to number of bytes to write (max 240) [out]pointer to number of bytes written
err	[out]pointer to returned error code

Note: All of the data byte to be written must be in the same sector. Only complete block(s) can be written. Be careful of sector trailer writing.

Return Value:

TRUE	successful
FALSE	failed, check error codes

Comment: This command is a combination of several single commands:
- Authenticate
- Write

Error Codes: Appendix A

ZBRGPMF_C_CreateValueBlock

Description: MIFARE Combined card command. Create a formatted value block. An automatic authentication can be performed or not before operation. An automatic write verification can be performed after operation.

Syntax:
```
int ZBRGPMF_C_CreateValueBlock (
                    HANDLE          hPrinter,
                    int             printerType,
                    byte            cardType,
                    byte            blockNumber,
                    byte            authentication,
                    byte            writeVerify,
                    long            value,
                    byte            data,
                    int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardType	[in] type of the current card: 0x00 = GEMEASY_8000 --> MIFARE 1K 0x02 = GEMCOMBI --> MIFARE 4K with automatic block value (Dual Interface Card) 0x03 = GEMEASY_32000 --> MIFARE 4K
blockNumber	[in] block number to authenticate: 0 to 62 with GEMEASY_8000 0 to 254 with GEMCOMBI or GEMEASY_32000
authentication	[in] automatic authentication control 0 = No Authentication 1 = Authentication KeyA 2 = Authentication KeyB
writeVerify	[in] perform write verification 0 = No Write Verification 1 = Perform Write Verification
value	[in] initial value to format the block (-2147483647 to + 2147483648)
data	[in] user data byte
err	[out]pointer to returned error code

Note: The block cannot be 60 to 7F for GEMCOMBI (MIFARE 4K w/auto block value).

Return Value:

TRUE	successful
FALSE	failed, check error codes

Comment: This command is a combination of several single commands:

- Authenticate
- Write

Value parameter is stored in Little-Endian Format

Error Codes: Appendix A

ZBRGPMF_C_ReadValue

Description: MIFARE Combined card command. Read the value of a formatted value block. An automatic authentication can be performed or not before operation.

Syntax:
```
int ZBRGPMF_C_ReadValue (
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                cardType,
                        byte                blockNumber,
                        byte                authentication,
                        long                *value,
                        byte                *data,
                        int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardType	[in] type of the current card: 0x00 = GEMEASY_8000 --> MIFARE 1K 0x02 = GEMCOMBI --> MIFARE 4K with automatic block value (Dual Interface Card) 0x03 = GEMEASY_32000 --> MIFARE 4K
blockNumber	[in] block number to authenticate: 0 to 62 with GEMEASY_8000 0 to 254 with GEMCOMBI or GEMEASY_32000
authentication	[in] automatic authentication control: 0 = No Authentication 1 = Authentication KeyA 2 = Authentication KeyB
value	[out]pointer to formatted value block read
data	[out]pointer to read user data byte read
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Comment: This command is a combination of several single commands:
- Authenticate
- Read

Error Codes: Appendix A

ZBRGPMF_C_SubtractValue

Description: MIFARE Combined card command. Subtract a value from a formatted value source block. The result is automatically transferred in a destination block. An automatic authentication can be performed or not before operations.

Syntax:
```
int ZBRGPMF_C_SubtractValue (
                    HANDLE          hPrinter,
                    int             printerType,
                    byte            cardType,
                    byte            blockNumber,
                    byte            authentication,
                    long            value,
                    byte            destinationBlock,
                    int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardType	[in] type of the current card: 0x00 = GEMEASY_8000 --> MIFARE 1K 0x02 = GEMCOMBI --> MIFARE 4K with automatic block value (Dual Interface Card) 0x03 = GEMEASY_32000 --> MIFARE 4K
blockNumber	[in] source block: 0 to 62 with GEMEASY_8000 0 to 254 with GEMCOMBI or GEMEASY_32000
authentication	[in] automatic authentication control: 0 = No Authentication 1 = Authentication KeyA 2 = Authentication KeyB
value	[in] value to be subtracted from source block (-2147483647 to + 2147483648)
destinationBlock	[in] destination block to store the result: 0 to 30 with GEMEASY_8000 0 to 254 with GEMCOMBI
err	[out]pointer to returned error code

Note: The source block and the destination block must be in the same sector.

Return Value:

TRUE	successful
FALSE	failed, check error codes

Comment: This command is a combination of several single commands:
- Authenticate
- Decrement
- Transfer

Error Codes: Appendix A

ZBRGPMF_C_AddValue

Description: MIFARE Combined card command. Add a value to a formatted value source block. The result is automatically transferred in a destination block. An automatic authentication can be performed or not before operations.

Syntax:
```
int ZBRGPMF_C_AddValue (
                    HANDLE          hPrinter,
                    int             printerType,
                    byte            cardType,
                    byte            blockNumber,
                    byte            authentication,
                    long            value,
                    byte            destinationBlock,
                    int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardType	[in] type of the current card: 0x00 = GEMEASY_8000 --> MIFARE 1K 0x02 = GEMCOMBI --> MIFARE 4K with automatic block value (Dual Interface Card) 0x03 = GEMEASY_32000 --> MIFARE 4K
blockNumber	[in] source block: 0 to 62 with GEMEASY_8000 0 to 254 with GEMCOMBI or GEMEASY_32000
authentication	[in] automatic authentication control: 0 = No Authentication 1 = Authentication KeyA 2 = Authentication KeyB
value	[in] value to be added from source block (-2147483647 to + 2147483648)
destinationBlock	[in] destination block to store the result: 0 to 30 with GEMEASY_8000 0 to 254 with GEMCOMBI
err	[out]pointer to returned error code

Note: The source block and the destination block must be in the same sector. The block cannot be 60 to 7F for GEMCOMBI (MIFARE 4K w/ auto block value).

Return Value:

TRUE	successful
FALSE	failed, check error codes

Comment: This command is a combination of several single commands:
- Authenticate
- Increment
- Transfer

Error Codes: Appendix A

ZBRGPMF_C_CopyValue

Description: MIFARE Combined card command. Copy a formatted value block to another block. An automatic authentication can be performed or not before operations.

Syntax:
```
int ZBRGPMF_C_CopyValue (
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                cardType,
                        byte                blockNumber,
                        byte                authentication,
                        byte                destinationBlock,
                        int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardType	[in] type of the current card: 0x00 = GEMEASY_8000 --> MIFARE 1K 0x02 = GEMCOMBI --> MIFARE 4K with automatic block value (Dual Interface Card) 0x03 = GEMEASY_32000 --> MIFARE 4K
blockNumber	[in] source block: 0 to 62 with GEMEASY_8000 0 to 254 with GEMCOMBI or GEMEASY_32000
authentication	[in] automatic authentication control: 0 = No Authentication 1 = Authentication KeyA 2 = Authentication KeyB
destinationBlock	[in] destination block to store the result: 0 to 30 with GEMEASY_8000 0 to 254 with GEMCOMBI
err	[out]pointer to returned error code

Note: The source block and the destination block must be in the same sector. The block cannot be 60 to 7F for GEMCOMBI (MIFARE 4K w/ auto block value).

Return Value:

TRUE	successful
FALSE	failed, check error codes

Comment: This command is a combination of several single commands:
- Authenticate
- Restore
- Transfer

Error Codes: Appendix A

ZBRGPMF_C_SetAccessConditions

Description: MIFARE Combined card command. Write the keys A, keys B, and access condition bits in a sector trailer (last block of a sector). An automatic authentication can be performed or not before operations.

Syntax:
```
int ZBRGPMF_C_SetAccessConditions(
                    HANDLE          hPrinter,
                    int             printerType,
                    byte            cardType,
                    byte            blockNumber,
                    byte            authentication,
                    byte            *keyA,
                    byte            accessBitsB0,
                    byte            accessBitsB1,
                    byte            accessBitsB2,
                    byte            accessBitsB3,
                    byte            *keyB,
                    int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardType	[in] type of the current card: 0x00 = GEMEASY_8000 --> MIFARE 1K 0x02 = GEMCOMBI --> MIFARE 4K with automatic block value (Dual Interface Card) 0x03 = GEMEASY_32000 --> MIFARE 4K
blockNumber	[in] source block: 0 to 63 with GEMEASY_8000, 0 to 255 with GEMCOMBI or GEMEASY_32000.
authentication	[in] automatic authentication control: 0 = No Authentication 1 = Authentication KeyA 2 = Authentication KeyB
keyA	[in] pointer to keyA to write in the sector trailer (6 bytes)
accessBitsB0	[in] access condition bits B0(0 to 7)
accessBitsB1	[in] access condition bits B1(0 to 7)
accessBitsB2	[in] access condition bits B2(0 to 7)
accessBitsB3	[in] access condition bits B3(0 to 7)
keyB	[in] pointer to keyB to write in the sector trailer (6 bytes)
err	[out]pointer to returned error code

Note: If the sector is a four-block sector B0 B1 B2 will be used for blocks 0, 1, and 2, and B3 for the sector trailer block 3.

If the sector is a sixteen-block sector, B0 B1 B2 will be used for blocks 0 to 4, 5 to 6, and 10 to 14 and B3 for the sector trailer block 15.

Be careful of sector trailer access condition B3. You can lock it.

Return Value:

TRUE	successful
FALSE	failed, check error codes

Comment: This command is a combination of several single commands:
- Authenticate
- Write

Error Codes: Appendix A

Access Conditions: See “Access Conditions” on page 233.

Reader Functions

ZBRGPMF_Reader_GetFirmware

Description: Read the version of the operating system implemented in the reader/writer.

Syntax:
```
int ZBRGPMF_Reader_GetFirmware(
                    HANDLE          hPrinter,
                    int             printerType,
                    byte            versionType
                    byte            *firmware
                    int             *len
                    int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
versionType	[in] fw version type to return: 1 = ROS Version 2 = OROS Version
firmware	[out]pointer to firmware version info buffer
len	[out]pointer to number of bytes received
err	[out]pointer to returned error code

Note: The reader returns:

- versionType = 1 (ROS)
  16 ASCII characters of the firmware version
  “ROS500-R3.40“ (4-space characters at the end)

- versionType = 2 (OROS)
  16 ASCII characters of the operating system version
  “OROS-R2.24 “ (6-space characters at the end)

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_Reader_GetID

Description: Retrieve the CL RC632 or MFRC531 9 byte product information: product type identification and product serial number.

Syntax:

```
int ZBRGPMF_Reader_GetID (
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                *productTypeID
                        int                 *productTypeIDSize
                        byte                *productSN
                        int                 *prodSNLen
                        int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
productTypeID	[out]pointer to product type ID buffer
productTypeIDSize	[in, out]: [in] pointer to productTypeID buffer size (min 5) [out] pointer to product type ID size
productSN	pointer to the product serial number buffer
prodSNLen	[in, out]: [in] pointer to productSN buffer size (min 4) [out] pointer to number of bytes read
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_Reader_GetModeAndGBPAddress

Description: Read the current operating mode and the current reader GBP address.

Syntax:
```
int ZBRGPMF_Reader_GetModeAndGBPAddress (
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                *mode,
                        byte                *gbpAddress,
                        int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
mode	[out]pointer to current reader mode
gbpAddress	[out]pointer to current GBP address
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_Reader_SetMode

Description: Set the reader's operating mode.

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

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
mode	[in] Reader operating mode 0x00 = Normal Mode (ISO14443A&B + MIFARE) 0x08 = MAD (ISO14443A + MIFARE) 0x0F = PayPass (ISO14443A&B)
err	[out]pointer to returned error code

Note: After printer power-up, mode 0 is selected.

Return Value:

TRUE	successful
FALSE	failed, check error codes

Comment: Normal Mode: Reader/writer is a slave device and is waiting for action.

MAD Mode: Reader/writer is a slave device that regularly scans the field, reads information stored into the smartcard using MAD format, and stores the information in an internal buffer.

PayPass Mode: Reader/writer will poll the field to search for PayPass smartcards. When a card is found, it will be automatically selected and is ready for payment operation.

Error Codes: Appendix A

ZBRGPMF_Reader_ReadEEPROM

Description: Read one byte of the EEPROM into the reader.
- The size of the EEPROM is 16 bytes length
- The first 8 bytes are used to configure the reader
- The 8 other bytes are free for use

Syntax:
```
int ZBRGPMF_Reader_ReadEEPROM (
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                address,
                        byte                *data,
                        int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
address	[in] address to read from - 0 to 15
data	[out]pointer to the EEPROM byte read
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_Reader_WriteEEPROM

Description: Write one byte of the EEPROM into the reader
- The size of the EEPROM is 16 bytes length
- The first 8 bytes are used to configure the reader
- The 8 other bytes are free for use

Syntax:
```
int ZBRGPMF_Reader_WriteEEPROM (
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                address,
                        byte                data,
                        int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
address	[in] address to read from - 0 to 15
data	[in] value to write to EEPROM
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_Reader_GetParameters

Description: Retrieve the reader internal parameters.

Syntax:
```
int ZBRGPMF_Reader_GetParameters (
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                *baudRateTypeA,
                        byte                *baudRateTypeB,
                        int                 *err)
```

Parameter:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
baudRateTypeA	[out]pointer to baud rates supported by reader in type A (1 byte)
baudRateTypeB	[out]pointer to baud rates supported by reader in type B (1 byte)
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_Reader_ActivateBuzzer

Description: This command is used to control the reader/writer buzzer.

Syntax:
```
int ZBRGPMF_Reader_ActivateBuzzer(
                        HANDLE          hPrinter,
                        int             printerType,
                        unsigned char   duration,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
duration	[in] 50ms based, 1 to 255 (50ms to 12750ms)
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGPMF_Reader_ChangeMode

Description: Modify the current operation mode.

Syntax:
```
int ZBRGPMF_Reader_ChangeMode(
                HANDLEhPrinter,
                int printerType,
                unsigned charmode,
                int *err)
```

Parameters:

hPrinter	[in] printer driver handle	
printerType	[in] printer type value, see Appendix B	
mode	[in] operation mode	
	0 = MODE_NORMAL	
	1 = MODE_WIEGAND_SERIAL_NUMBER_MASTER_CYCLIC_AUTO_READ	
	2 = MODE_WIEGAND_SERIAL_NUMBER_MASTER_ONE_TIME_AUTO_READ	
	5 = MODE_CYCLIC_AUTO_READ	
	6 = MODE_ONE_TIME_AUTO_READ	
	7 = MODE_SLAVE_ONE_TIME_AUTO_READ	
	8 = MODE_MAD_SLAVE_ONE_TIME_AUTO_READ	
	9 = MODE_MAD_WIEGAND_ONE_TIME_AUTO_READ	
	15 = MODE_SLAVE_AUTOMATIC_CARD_RECOGNITION	
err	[out]pointer to returned error code	

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGPMF_Reader_ControlLeds

Description: This command is used to control the reader/writer LED activity.

Syntax:
```
int ZBRGPMF_Reader_ControlLeds(
                        HANDLE hPrinter,
                        intprinterType,
                        unsigned charctrlStatePhase1,
                        unsigned char ctrlStatePhase2,
                        unsigned chardurationPhase1,
                        unsigned chardurationPhase2,
                        unsigned charmode,
                        unsigned charrepetition
                        int *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
ctrlPhaseState1	[in] state of phase 1 led 0 = Led Off 1 = Green Led On 2 = Red Led On 3 = Green and Red Led On
ctrlPhaseState2	[in] state of phase 2 led 0 = Led Off 1 = Green Led On 2 = Red Led On 3 = Green and Red Led On
durationPhase1	[in] 50ms based, 0 to 255 (0ms to 12750ms)
durationPhase2	[in] 50ms based, 1 to 255 (50ms to 12750ms)
mode	[in] led mode 0 = Leds always off 16 = Leds use phases 1 and 2 32 = Leds always on
repetition	[in] repetition mode 0 = infinite repetition phases 1 and 2 1-15 = repetition of phase 1 and 2
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGPMF_Reader_OpenCaseDetection

Description: Detects the reader casing being opened. After reception of this command, the reader will be locked as soon as the casing is opened or immediately if casing is already opened. While the reader is locked, only the Command: GCLSRAPI_Reader_OpenCaseDetection with the parameter: ACKNOWLEDGE_OCD will be executed.

Syntax:

```
int ZBRGPMF_Reader_OpenCaseDetection(
                        HANDLE          hPrinter,
                        int             printerType,
                        unsigned char   action,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
action	[in] open case detection action 1 = activate open case detection 2 = acknowledge open case detection
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGPMF_Reader_SetDelay

Description: Set a delay between the reception of the command and the reader's answer.

Syntax:
```
int ZBRGPMF_Reader_SetDelay(
                        HANDLEhPrinter,
                        intprinterType,
                        unsigned charduration,
                        int*err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
duration	[in] 1ms based, 1 to 255 (1ms to 255ms)
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

RF Functions

ZBRGPMF_RF_Control

Description: Turn ON, OFF, or RESET reader's RF field.

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

Parameter:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
mode	[in] pointer controls the RF state: 1 = RF On 2 = RF Off 3 = RF Reset
err	[out]pointer to returned error code

Note: ON - After the command is executed, the card will be in the Idle state if field was previously off.

OFF – After the command is executed, the card will be in the Power Off state.

RESET – After the command is executed, the card will be in the Idle state.

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_RF_ChangeModulationType

Description: This command is used to change the RF modulation type.

Syntax:
```
int ZBRGPMF_RF_ChangeModulationType (
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                modType,
                        int                 *err)
```

Parameter:

	hPrinter	[in] printer driver handle
	printerType	[in] printer type value, see Appendix B
	modType	[in] pointer controls the RF state: 0 = Type A 1 = Type B
	err	[out]pointer to returned error code

Return Value:

	TRUE	successful
	FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_RF_ReadModulationType

Description: This command is used to read the RF modulation type.

Syntax:
```
int ZBRGPMF_RF_ReadModulationType (
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                *modType,
                        int                 *err)
```

Parameter:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
modType	[in] pointer to returned modulation type: 0 = Type A 1 = Type B
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

14443A Functions

ZBRGPMF_ISO14443_3_A_RequestA

Description: This command is used to detect ISO14443A cards in the field that are not previously found.

Syntax:
```
int ZBRGPMF_ISO14443_3_A_RequestA (
                        HANDLE                  hPrinter,
                        int                     printerType,
                        byte                    card,
                        short                   *ATQA,
                        int                     *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
card	[in] First card or Next card: 0 = First Card 1 = Next Card
ATQA	[out]pointer to returned card ATQA (1 byte)
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_ISO14443_3_A_Anticollision

Description: This command is used to retrieve the serial number from an ISO14443A card in the field.

Syntax:
```
int ZBRGPMF_ISO14443_3_A_Anticollision (
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                cascadeLevel,
                        byte                *cascadeLevelSN,
                        int                 cascadeLevelSNLen,
                        int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cascadeLevel	[in] Cascade Level: 0 = CASCADE_LEVEL_NOT_SPECIFIED 1 = CASCADE_LEVEL_1 (4-byte serial number) 2 = CASCADE_LEVEL_2 (7-byte serial number) 3 = CASCADE_LEVEL_3 (10-byte serial number)
cascadeLevelSN	[out]pointer to card serial number buffer
cascadeLevelSNLen	[in] size of cascadeLevelSN buffer (mim 4 bytes)
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_ISO14443_3_A_Select

Description: This command is used to select one individual ISO14443A card for further operations as anticollision with higher cascade level, authentication, and memory related operations.

Syntax:
```
int ZBRGPMF_ISO14443_3_A_Select (
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                cascadeLevel,
                        byte                *cascadeLevelSN,
                        int                 cascadeLevelSNLen,
                        byte                *SAK,
                        int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cascadeLevel	[in] Cascade Level: 0 = CASCADE_LEVEL_NOT_SPECIFIED 1 = CASCADE_LEVEL_1 (4-byte serial number) 2 = CASCADE_LEVEL_2 (7-byte serial number) 3 = CASCADE_LEVEL_3 (10-byte serial number)
cascadeLevelSN	[out]pointer to card serial number buffer
cascadeLevelSNLen	[in] size of serial number returned (minimum = 4)
SAK	[out]pointer to Select Acknowledge Type A of card
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_ISO14443_3_A_Halt

Description: Performs an ISO 14443-A HALT command for the selected card.

Syntax:
```
int ZBRGPMF_ISO14443_3_A_Halt (
                        HANDLE          hPrinter,
                        int             printerType,
                        int             *err)
```

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

Combined 14443A Functions

ZBRGPMF_ISO14443_3_A_GetCard

Description: This command is used to search for the first or next ISO14443A-3 card in the field.

Syntax:
```
int ZBRGPMF_ISO14443_3_A_GetCard(
                        HANDLE              hPrinter,
                        int                 printerType,
                        sCARD_AND_TIMEOUT   *cardAndTimeout,
                        sGET_CARD_A         *getCardAInfo,
                        int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardAndTimeout	[in] pointer to structure (see definition) indicating first or next card and whether the timeout is used

```
typedef struct S_CARD_AND_TIMEOUT
{
    //0 = First Card
    //1 = Next Card
    byte ucCard;
    //0 = Timeout Not Specified
    //1 = Timeout Specified
    byte ucIsTimeoutSpecified;
    // Only taken into account in case
    timeout set as specified byte
    ucTimeout_50msBased;
}sCARD_AND_TIMEOUT;
```

getCardAInfo	[out]pointer to structure (see definition) Type A card information

```
typedef struct S_GET_CARD_A
{
    //4 = One cascade level length
    //7 = Two cascade level length
    //10 = Three cascade level length
    int iSerialNumberSize;
    byte *pucSerialNumber;
    short usATQA;
    byte ucSAK;
}sGET_CARD_A;
```

err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Comment: This command is a combination of several single commands:
- Halt A
- Request A
- Anticollision
- Select

Error Codes: Appendix A

Structure Definitions: See “Structure Definitions” on page 231.

ZBRGPMF_ISO14443_3_A_RequestAllSelectA

Description: This single command uses a specified serial number in which only the specified ISO14443A-3 card will respond.

Syntax:

```
int ZBRGPMF_ISO14443_3_A_RequestAllSelectA (
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                *serialNumber,
                        int                 serialNumberLen,
                        byte                *SAK,
                        int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
serialNumber	[in] pointer to serial number of card to select
serialNumberLen	[in] size of serialNumber buffer: 4 = One cascade level length 7 = Two cascade level length 10 = Three cascade level length
SAK	[out]pointer to acknowledge Type A (SAK) of card
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Comment: This command is a combination of several single commands:

- Halt A
- Request A
- Select A

Error Codes: Appendix A

ZBRGPMF_ISO14443_3_A_GetCardA_T_CL

Description: This command is used to search for the first or next ISO14443A-4 T=CL card in the field.

Syntax:
```
int ZBRGPMF_ISO14443_3_A_GetCardA_T_CL (
                        HANDLE              hPrinter,
                        int                 printerType,
                        sCARD_AND_TIMEOUT   *cardAndTimeout,
                        sGET_CARD_A         *getCardAInfo,
                        sGET_CARD_A_T_CL    *getCardATclInfo,
                        byte                PPSBaudRates
                        int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardAndTimeout	[in] pointer to structure (see definition) indicating first or next card and whether the timeout is used
getCardAInfo	[out]pointer to structure (see definition) giving Type A card information
getCardATclInfo	[out]pointer to structure (see definition) giving Type A T=CL card information
PPSBaudRates	[in] Mask bit of allowed baudrates to be used by PPS, same coding as ATS TA1 parameter
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Comment: This command is a combination of several single commands:

- Request A
- Anticollision
- Select
- Request for answer to select
- Protocol parameters selection

Error Codes: Appendix A

Structure Definitions: See “Structure Definitions” on page 231.

ZBRGPMF_ISO14443_3_A_RequestAllSelectA_T_CL

Description: This single command uses a specified serial number in which only the specified ISO14443A-4 card will respond.

Syntax:
```
int ZBRGPMF_ISO14443_3_A_RequestAllSelectA_T_CL(
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                PPSBaudRates,
                        byte                *serialNumber,
                        int                 serialNumberLen,
                        sGET_CARD_A_T_CL    *getCardATclInfo,
                        byte                *SAK,
                        int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
PPSBaudRates	[in] Mask bit of allowed baudrates to be used by PPS, same coding as ATS TA1 parameter
serialNumber	[in] pointer to card serial number buffer
serialNumberLen	[in] size of serialNumber buffer: 4 = One cascade level length 7 = Two cascade level length 10 = Three cascade level length
getCardATclInfo	[out]pointer to structure (see definition) giving Type A T=CL card information
SAK	[out]pointer to acknowledge Type A (SAK) of card
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Comment: This command is a combination of several single commands:
- Halt A
- Request All A
- Select

Error Codes: Appendix A

Structure Definitions: See “Structure Definitions” on page 231.

14443_4_A Functions

ZBRGPMF_ISO14443_4_A_RequestForAnswerToSelect

Description: This command is used to start ISO14443A protocol activation from previously selected cards if an Answer To Select (ATS) is available.

Syntax:

```
int ZBRGPMF_ISO14443_4_A_RequestForAnswerToSelect (
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                CID,
                        byte                *ATS,
                        int                 *ATSSize,
                        int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
CID	[in] desired card identifier from CID_MIN(0) to CID_MAX(14)
ATS	[out]pointer to Answer To Select buffer
ATSSize	[in, out]: [in]pointer to size of ATS buffer [out]pointer to number of bytes read
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_ISO14443_4_A_ProtocolParameterSelection

Description: This command is used to change ISO14443A cards parameters if supported by the card.

Syntax:
```
int ZBRGPMF_ISO14443_4_A_ProtocolParameterSelection (
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                CID,
                        byte                DSI,
                        byte                DRI,
                        int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
CID	[in] Desired card identifier from CID_MIN(0) to CID_MAX(14).
DSI	[in] Card Baud Rate selection: 106 kbps = 0x00 212 kbps = 0x01 424 kbps = 0x02 848 kbps = 0x03
DRI	[in] Reader Baud rate selection: 106 kbps = 0x00 212 kbps = 0x01 424 kbps = 0x02 848 kbps = 0x03
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

14443_4_A_B Functions

ZBRGPMF_ISO14443_4_A_B_Exchange_T_CL

Description: This command is used to exchange data with a card that is in the Active state using T=CL protocol as defined in ISO14443A&B-4.

Syntax:
```
int ZBRGPMF_ISO14443_4_A_B_Exchange_T_CL (
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                CID,
                        byte                NAD,
                        int                 commandLength,
                        byte                *command,
                        byte                *response,
                        int                 *responseSize,
                        int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
CID	[in] desired card identifier from CID_MIN(0) to CID_MAX(14)
NAD	[in] Node Address Logical Connection ISO7816-3 compliant, SAD and DAD between 0 and 7
commandLength	[in] Length of the Command to send to the card. No constraint at this API level. See reader and card documentations to get the maximum available length
command	[in] pointer to the command to send to the card
response	[out]pointer to data returned from the card
responseSize	[in, out]: [in] length of response buffer, min size = 1 [out]length of data returned from the card
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_ISO14443_4_A_B_Deselect

Description: This command is used to deactivate a card that is in the Active state.

Syntax:
```
int ZBRGPMF_ISO14443_4_A_B_Deselect (
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            CID,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
CID	[in] desired card identifier from CID_MIN(0) to CID_MAX(14)
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_ISO14443_4_A_B_Poll_T_CL_Card_Removed

Description: This command waits for a T=CL card in the reader field. After selecting a T=CL card you must exchange data with the card to use this command.

Syntax:
```
int ZBRGPMF_ISO14443_4_A_B_Poll_T_CL_Card_Removed (
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                CID,
                        int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
CID	[in] desired card identifier from CID_MIN(0) to CID_MAX(14)
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_ISO14443_4_A_B_Mode15_GetStatus

Description: This command is used to get the reader status in Mode 15.

Syntax:
```
int ZBRGPMF_ISO14443_4_A_B_Mode15_GetStatus (
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                *response,
                        int                 *responseSize
                        int                 *err)
```

Parameters:

	hPrinter	[in] printer driver handle
	printerType	[in] printer type value, see Appendix B
	response	[out]pointer to the data returned from the reader
	responsesize	[in, out]: [in] pointer to length of response buffer, min size = 1, max size = 510 [out]pointer to length of data returned from reader
	err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_GEMCORECARDI_Card_Exchange

Description: Performs a TPDU or an APDU exchange with a card.

Syntax:
```
int ZBRGPMF_GEMCORECARDI_Card_Exchange (
                        HANDLE          hPrinter,
                        int             printerType,
                        unsigned char   cardInterface,
                        unsigned char   cardExchangeType,
                        unsigned char   *cardCommand,
                        int             cardCommandSize,
                        unsigned char   *cardResponse,
                        int             *cardResponseSize,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardInterface	[in] target card interface MAIN_CARD_INTERFACE = 1 AUXILIARY_CARD_INTERFACE = 0
cardExchangeType	[in] Card Exchange Command Code 0 = TPDU, data from card to reader 1 = TPDU, data from reader to card 2 = APDU, possible data in both directions
cardCommand	[in] pointer to Card Command: CLA INS P1 P2...
cardResponse	[out]pointer to card response
cardResponseSize	[in/out]pointer to cardResponse size In : Size of the cardResponse buffer (min size = 1) Out: cardResponse length.
err	[out]pointer to reader response status GCLSR_STATUS_NO_ERROR or reader status error

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGPMF_GEMCORECARDI_Exchange_IFSD

Description: This command powers up and resets a card, automatically followed if T=1 protocol in use by an IFSD exchange with the card specifying the new Information Field Size interface Device.

Syntax:
```
int ZBRGPMF_GEMCORECARDI_Exchange_IFSD(
                    HANDLE hPrinter,
                    intprinterType,
                    unsigned charcardInterface,
                    unsigned char*IFSD,
                    int*err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardInterface	[in] target card interface MAIN_CARD_INTERFACE = 1 AUXILIARY_CARD_INTERFACE = 0
IFSD	[in/out]pointer to ISO 7816-3 compliant Info Field Size interface device parameter From 1 to 254 In: IFSD Request Out: IFSD acknowledged by the reader
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGPMF_GEMCORECARDI_Exchange_PPS

Description: This command may (if PPS not specified as Manual) power up and reset a card, automatically followed by the specified Protocol and Parameters Selection.

Syntax:

```
int ZBRGPMF_GEMCORECARDI_Exchange_PPS(
                        HANDLE          hPrinter,
                        int             printerType,
                        unsigned char   cardInterface,
                        unsigned char   CFG,
                        unsigned char   *PPSRequest,
                        int             PPSRequestSize,
                        unsigned char   *PPSResponse,
                        int             *PPSResponseSize,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardInterface	[in] target card interface MAIN_CARD_INTERFACE = 1 AUXILIARY_CARD_INTERFACE = 0
CFG	[in] configuration byte indicating parameters of the card activation; see Gemcore Reference Manual to get a complete and detailed desciption of possible values
PPSRequest	[in] pointer to ISO 7816-3 compliant PPS frame, but without the first byte PPSS = 0xFF automatically added by the reader.
PPSRequestSize	[in] length of PPSRequest parameter 0 to 6
PPSResponse	[out]pointer to card PPS Response buffer
PPSResponseSize	[in/out]pointer to PPSResponse size In : Size of PPSResponse buffer (min = 1) Out: PPS response length
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGPMF_GEMCORECARDI_PowerDown

Description: This command is used to power down the card.

Syntax:
```
int ZBRGPMF_GEMCORECARDI_PowerDown(
                        HANDLE          hPrinter,
                        int             printerType,
                        unsigned char   cardInterface,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardInterface	[in] Target card interface MAIN_CARD_INTERFACE = 1 AUXILIARY_CARD_INTERFACE = 0
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGPMF_GEMCORECARDI_PowerUp

Description: This command is used to power up the card.

Syntax:
```
int ZBRGPMF_GEMCORECARDI_PowerUp(
                        HANDLE          hPrinter,
                        int             printerType,
                        unsigned char   cardInterface,
                        unsigned char   useCFG,
                        unsigned char   CFG,
                        unsigned char   *ATR,
                        int             *ATRSize,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardInterface	[in] target card interface MAIN_CARD_INTERFACE = 1 AUXILIARY_CARD_INTERFACE = 0
useCFG	[in] boolean indicating whether the CFG parameter is used or not: TRUE = used FALSE = not used
CFG	[in] configuration byte indicating parameters of the card activation; see Gemcore Reference Manual to get a complete and detailed desciption of possible values.
ATR	[in] pointer to card Answer To Reset
ATRSize	[in/out] pointer to pucATR size In : Size of the ATR buffer (min = 1) Out: ATR length.
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGPMF_GEMCORECARDI_SetCurrentSAM

Description: This command selects the auxiliary card number.

Syntax:
```
int ZBRGPMF_GEMCORECARDI_SetCurrentSAM(
                        HANDLE          hPrinter,
                        int             printerType,
                        unsigned char   samNumber,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
samNumber	[in] represents the auxiliary card number 1 to 4
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRGPMF_GEMCORECARDI_Status

Description: This command is used to obtain the status of the main card interface or of the auxiliary card. It returns information indicating:
- The type of card currently used
- Card presence
- The power supply value
- The card power status
- The communication protocol (T=0 or T=1)
- The speed parameters between the card and the reader

Syntax:
```
int ZBRGPMF_GEMCORECARDI_Status(
                    HANDLE          hPrinter,
                    int             printerType,
                    unsigned char   cardInterface,
                    sCARD_STATUS    *cardStatus,
                    int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardInterface	[in] Target card interface MAIN_CARD_INTERFACE = 1 AUXILIARY_CARD_INTERFACE = 0
cardStatus	[out]pointer to Card Status structure (see definition)
err	[out]pointer to returned error code

```
typedef struct S_CARD_STATUS
{
  unsigned char ucSTAT;
  unsigned char ucCardTypeSelection;
  unsigned char ucFiDi;
  unsigned char ucExtraGuardTime;
  union
  {
    sSTATUS_T0 sStatusT0;
    sSTATUS_T1 sStatusT1;
  };
}sCARD_STATUS;
```

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

14443B Functions

ZBRGPMF_ISO14443_3_B_RequestB

Description: This command is used to probe the field for ISO14443B cards that are not previously found.

Syntax:
```
int ZBRGPMF_ISO14443_3_B_RequestB (
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                card,
                        byte                AFI,
                        byte                numberOfSlots,
                        sATQB               *ATQB,
                        int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
card	[in] First card or Next card 0 = First Card 1 = Next Card
AFI	[in] application family identifier
numberOfSlots	[in] number of slots for anticolision process: 1, 2, 4, 8, 16
ATQB	[out]pointer to returned ATQB structure
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

Structure Definitions: See “Structure Definitions” on page 231.

ZBRGPMF_ISO14443_3_B_SlotMarker

Description: This command is used to send slot markers commands to the ISO14443B card to define the start of each timeslot required by the anticollision process.

Syntax:
```
int ZBRGPMF_ISO14443_3_B_SlotMarker (
                        HANDLE              hPrinter,
                        int                 printerType,
                        byte                slotNumber,
                        sATQB               *ATQB,
                        int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
slotNumber	[in] slot number, from 2 to 16
ATQB	[out]pointer to returned ATQB structure
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

Structure Definitions: See “Structure Definitions” on page 231.

ZBRGPMF_ISO14443_3_B_Attribute

Description: This command is used to select one individual ISO14443B card and start T=CL protocol activation. Only the card in Ready declared state and with the corresponding PUPI will answer.

Syntax:

```
int ZBRGPMF_ISO14443_3_B_Attribute (
                        HANDLE                      hPrinter,
                        int                         printerType,
                        sPseudoUniquePICCIdentifier *PUPIId,
                        sProtocolInfo               *protocolInfo,
                        byte                        CID,
                        byte                        bitRate,
                        byte                        *maxLenBufInd_CID,
                        int                         *err)
```

Parameters

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
PUPIId	[in] pointer to sPseudoUniquePICCIdentifier structure
protocolInfo	[in] pointer to protocolInfo structure
CID	[in] desired card identifier: 0 to 14
bitRate	[in] desired bit rates
maxLenBufInd_CID	[out]pointer to answer to attribute (1 byte)
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

Structure Definitions: See “Structure Definitions” on page 231.

ZBRGPMF_ISO14443_3_B_Halt

Description: Performs a T=CL block exchange according to ISO 14443-B-4

Syntax:
```
int ZBRGPMF_ISO14443_3_B_Halt (
                        HANDLE                      hPrinter,
                        int                         printerType,
                        sPseudoUniquePICCIdentifier *PUPIId,
                        int                         *err)
```

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

Structure Definitions: See “Structure Definitions” on page 231.

Combined 14443B Function

ZBRGPMF_ISO14443_3_B_GetCard

Description: This command is used to search for ISO14443B-3 card in the field using appropriate anticollision procedure.

Syntax:
```
int ZBRGPMF_ISO14443_3_B_GetCard (
                        HANDLE              hPrinter,
                        int                 printerType,
                        sCARD_AND_TIMEOUT   *cardAndTimeout,
                        byte                AFI,
                        byte                bitrates,
                        sGET_CARD_B         *getCardBInfo,
                        int                 *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
cardAndTimeout	[in] pointer to structure (see definition) indicating first or next card and whether the timeout is used
AFI	[in] application family identifier
bitRates	[in] mask bit of allowed baudrates to be used by the Attrib, same coding as the Bit_Rate_capability of Protocol Info
getCardBInfo	[out]pointer to structure (see definition) giving Type B card info
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Comment: This command is a combination of several single commands:

- RF Reset
- Request All B
- Slot Marker
- Attribute

Error Codes: Appendix A

Structure Definitions: See “Structure Definitions” on page 231.

Transparent Functions

ZBRGPMF_TransparentExchange

Description: This command is used to transfer data transparently to the smart card. Use ZBRGPMF_TransparentExchangeTimeout to define a timeout value.

Syntax:
```
int ZBRGPMF_TransparentExchange(
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            *dataIn,
                        int             dataInSize,
                        byte            *dataOut,
                        int             *dataOutSize,
                        int             *respSize,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
dataIn	[in] pointer to data to send to the card
dataInSize	[in] length of the command: 1 to 256 bytes
dataOut	[out]pointer to response buffer
dataOutSize	[in] pointer to size of dataOut
respSize	[out]pointer to size of response buffer
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRGPMF_TransparentExchangeTimeout

Description: This command defines a timeout value for the ZBRGPMF_TransparentExchange command.

Syntax:
```
int ZBRGPMF_TransparentExchangeTimeout (
                        HANDLE          hPrinter,
                        int             printerType,
                        long            timeout,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
timeout	[in] 3-byte value - 0x000000 to 0x3FC000 T = timeout * 128 / 13.56 microsecond Max = 39.47 seconds (0x3FC000)
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBR_TL_SendReceive

Description: Sends the GBP command and retrieves the reader response.

Syntax:
```
int ZBR_TL_SendReceive (
                        unsigned short intusiLnCmd,
                        unsigned char     *prgucCmd,
                        unsigned short intpusiLnResp,
                        unsigned char     *prgucResp)
```

Parameters:

usiLnCmd	[in] length of the command
prgucCmd	[in] pointer to buffer with the command
pusiLnResp	[out]length of the response
prgucResp	[out[pointer to buffer with response

Return Value:
```
NO_ERR
ERR_LRC
ERR_SEQUENCE_NUMBER
ERR_NACK
ERR_READER_MUTE
```

Structure Definitions

RF.H

```
RF_ON                                   1
RF_OFF                                  2
RF_RESET                                3
```

GemCoreCardI.H

```
MAIN_CARD_INTERFACE                     1
AUXILIARY_CARD_INTERFACE                0

STAT_PROTOCOL_MASK_BIT                  0x08
STAT_PROTOCOL_T0                        0x00
STAT_PROTOCOL_T1                        0x08
```

ISO14443_3.H

```
FIRST_CARD                              0
NEXT_CARD                               1
FIRST_CARD_T_CL                         2
NEXT_CARD_T_CL                          3

TIMEOUT_NOT_SPECIFIED                   0
TIMEOUT_SPECIFIED                       1

typedef struct S_CARD_AND_TIMEOUT
{
    byte ucCard;
    byte ucIsTimeoutSpecified;
    byte ucTimeout_50msBased;
}sCARD_AND_TIMEOUT;
```

ISO14443_3_A.H

```
ATS_MIN_LENGTH                          1
ATS_MAX_LENGTH                          254

CASCADE_LEVEL_NOT_SPECIFIED             0
CASCADE_LEVEL_1                         1 ( 4-byte serial number)
CASCADE_LEVEL_2                         2 ( 7-byte serial number)
CASCADE_LEVEL_3                         3 (10-byte serial number)
ONE_CASCADE_LEVEL_SERIAL_NUMBER_SIZE    4

typedef struct S_GET_CARD_A
{
    /* IN: Size of the following buffer */
    /* OUT: Serial Number length */
    int iSerialNumberSize;
    /* Can be from 1 to 3 Cascade Level */
    byte *pucSerialNumber;
    short usATQA;
    byte ucSAK;
}sGET_CARD_A;


typedef struct S_GET_CARD_A_T_CL
{
    byte ucFSDI;
    byte ucCID;
    /* IN: Size of the following buffer */
    /* OUT: ATS length */
    int iATSSize;
    byte *pucATS;
    byte ucDSIe;
    byte ucDRIe;
}sGET_CARD_A_T_CL;
```

ISO14443_3_B.H

```
BIT_RATE_CAPABILITY_PICC_106_BOTH_DIRECTIONS_ONLY  0x00
BIT_RATE_CAPABILITY_SAME_IN_BOTH_DIRECTIONS        0x80
BIT_RATE_CAPABILITY_PICC_TO_PCD_212                0x10
BIT_RATE_CAPABILITY_PICC_TO_PCD_424                0x20
BIT_RATE_CAPABILITY_PICC_TO_PCD_847                0x40
BIT_RATE_CAPABILITY_PCD_TO_PICC_212                0x01
BIT_RATE_CAPABILITY_PCD_TO_PICC_424                0x02
BIT_RATE_CAPABILITY_PCD_TO_PICC_8470x04


#DEFINE PUPI_SIZE 4
typedef struct S_PSEUDO_UNIQUE_PICC_IDENTIFIER
{
   byte ucByte[PUPI_SIZE];
}sPseudoUniquePICCIdentifier;


typedef struct S_APPLICATION_DATA
{
   byte ucApplicationFamilyIdentifier;
   byte ucCRC_B_AID[2];
   byte ucNumbersOfApplications;
}sApplicationData;


typedef struct S_PROTOCOL_INFO
{
   byte ucBitRateCapability;
   byte ucMaxFrameSize_ProtocolType;
   byte ucFrameWaitingTimeInteger_ApplicationDataCoding_FrameOption;
}sProtocolInfo;


typedef struct S_ATQB
{
   byte ucFirstByte;
   sPseudoUniquePICCIdentifier sPUPIId;
   sApplicationData sAppData;
   sProtocolInfo sProInfo;
}sATQB;


typedef struct S_GET_CARD_B
{
   sATQB sCurATQB;
   byte ucATTRIB;
   byte ucBITRATE;
}sGET_CARD_B;
```

Access Conditions

Data Block

Access bits for the data blocks are defined as Never, KeyA or KeyB.

Access Bits			Access Condition				
C1	C2	C3	Read	Write	Increment	Decrement Transfer Restore	Comments
0	0	0	KeyA | KeyB	KeyA | KeyB	KeyA | KeyB	KeyA | KeyB	A or B All Function Memory Block
0	0	1	KeyA | KeyB	Never	Never	KeyA | KeyB	A or B Read/ Subtract Value Block
0	1	0	KeyA | KeyB	Never	Never	Never	A or B Read Only Memory Block
0	1	1	KeyB	KeyB	Never	Never	B Read/Write Memory Block
1	0	0	KeyA | KeyB	KeyB	Never	Never	A or B Read and B Write Memory Block
1	0	1	KeyB	Never	Never	Never	B Read Only Memory Block
1	1	0	KeyA | KeyB	KeyB	KeyB	KeyA | KeyB	A or B Read/ Subtract and B Write/Increment Memory Block
1	1	1	Never	Never	Never	Never	Locked Block Access never Allowed

Sector Trailer

Access Bits			Access Condition						
			Authenticate KeyA		Access Bits		Authenticate KeyB		
C1	C2	C3	Read	Write	Read	Write	Read	Write	Comments
0	0	0	Never	KeyA	KeyA	Never	KeyA	KeyA	KeyB May Be Read
0	0	1	Never	KeyA	KeyA	KeyA	KeyA	KeyA	KeyB May Be Read (transport config)
0	1	0	Never	Never	KeyA	Never	KeyA	Never	KeyB May Be Read
0	1	1	Never	KeyB	KeyA | KeyB	KeyB	Never	KeyB	
1	0	0	Never	KeyB	KeyA | KeyB	Never	Never	KeyB	
1	0	1	Never	Never	KeyA | KeyB	KeyB	Never	Never	
1	1	0	Never	Never	KeyA | KeyB	Never	Never	Never	
1	1	1	Never	Never	KeyA | KeyB	Never	Never	Never	

Shaded areas are access conditions where KeyB is readable and can be used for data.

MIFARE Key Management

Up to 16 key A and 16 key B can be stored in the reader/writer using the *Load Keys* command.

Key A and key B are stored in Reader/Writer EEPROM's non-volatile memory locations named *Keys Sectors*.

There are 40 sectors in the card, but only 16 are in the reader/writer. Each reader/writer key sector is used to authenticate either two or three sectors in the card:

Key Management Table

Reader/Writer Key Sector	Card Sector 0 - 15	Card Sector 16 - 31	Card Sector 32 - 39
Sector 0	Sector 0	Sector 16	Sector 32
Sector 1	Sector 1	Sector 17	Sector 33
Sector 2	Sector 2	Sector 18	Sector 34
Sector 3	Sector 3	Sector 19	Sector 35
Sector 4	Sector 4	Sector 20	Sector 36
Sector 5	Sector 5	Sector 21	Sector 37
Sector 6	Sector 6	Sector 22	Sector 38
Sector 7	Sector 7	Sector 23	Sector 39
Sector 8	Sector 8	Sector 24	-
Sector 9	Sector 9	Sector 25	-
Sector 10	Sector 10	Sector 26	-
Sector 11	Sector 11	Sector 27	-
Sector 12	Sector 12	Sector 28	-
Sector 13	Sector 13	Sector 29	-
Sector 14	Sector 14	Sector 30	-
Sector 15	Sector 15	Sector 31	-

For an example, read/write key sector 7 holds the keys needed to authenticate sector 7, sector 23, and sector 39 of the smart card.

Note: This command does not perform any access to the smart card.

EEPROM Management

An EEPROM memory is available to the user to store information.

The size of the EEPROM memory is 16 bytes, but the first 8 bytes are reserved for the reader/writer as shown in the following table:

EEPROM Management Table

EEPROM User Address	Memory Content	Comment
0h to 7h	Reserved for reader/writer configuration	Not available to users
8h	Free	Available to users
9h	Free	Available to users
Ah	Free	Available to users
Bh	Free	Available to users
Ch	Free	Available to users
Dh	Free	Available to users
Eh	Free	Available to users
Fh	Free	Available to users

The EEPROM memory can be accessed using the ZBRGPMF_Reader_ReadEEPROM or ZBRGPMF_Reader_WriteEEPROM command.

Answer To Request, Type A (ATQA)

All MIFARE® chips respond to the ISO/IEC 14443A-3 ‘REQA’ (Request Command, Type A) with the appropriate ATQA (Answer To Request, Type A). The ATQA is two bytes long and contains the information as mapped in the table below:

ATQA Table*

	MSB ATQA								LSB ATQA							
Bit number	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
ATQA bit values defined in the ISO/IEC 14443A-3																
Coding of ATQA according to ISO/IEC 14443A-3	RFU[1]				Proprietary Coding				UID size bit frame		RFU[1]	Bit frame anticollision				
Proprietary								1								
Proprietary							1									
Proprietary						1										
Single UID									0	0						
Double UID									0	1						
Triple UID									1	0						
RFU									1	1						
Bit Frame Anticollision supported												1	0	0	0	0
Bit Frame Anticollision supported												0	1	0	0	0
Bit Frame Anticollision supported												0	0	1	0	0
Bit Frame Anticollision supported												0	0	0	1	0
Bit Frame Anticollision supported												0	0	0	0	1
ATQA response values of different MIFARE® chips																
MIFARE® ultralight (0x0044)	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0
MIFARE® 1K (0x0004)	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
MIFARE® 4K (0x0002)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
MIFARE® DESFire (0x0344)	0	0	0	0	0	0	1	1	0	1	0	0	0	1	0	0
MIFARE® ProX (0xXX08)	0	0	0	0	0	x^2	x^2	x^2	0	0	0	0	1	0	0	0
MIFARE® ProX (0xXX04)	0	0	0	0	0	x^2	x^2	x^2	0	0	0	0	0	1	0	0
MIFARE® ProX (0xXX02)	0	0	0	0	0	x^2	x^2	x^2	0	0	0	0	0	0	1	0
MIFARE® ProX (0xXX48)	0	0	0	0	0	x^2	x^2	x^2	0	1	0	0	1	0	0	0
MIFARE® ProX (0xXX44)	0	0	0	0	0	x^2	x^2	x^2	0	1	0	0	0	1	0	0
MIFARE® ProX (0xXX42)	0	0	0	0	0	x^2	x^2	x^2	0	1	0	0	0	0	1	0
SmartMX xD(T) (0xXX08)	0	0	0	0	0	x^2	x^2	x^2	0	0	0	0	1	0	0	0
SmartMX xD(T) (0xXX04)	0	0	0	0	0	x^2	x^2	x^2	0	0	0	0	0	1	0	0
SmartMX xD(T) (0xXX02)	0	0	0	0	0	x^2	x^2	x^2	0	0	0	0	0	0	1	0
SmartMX xD(T) (0xXX48)	0	0	0	0	0	x^2	x^2	x^2	0	1	0	0	1	0	0	0
SmartMX xD(T) (0xXX44)	0	0	0	0	0	x^2	x^2	x^2	0	1	0	0	0	1	0	0
SmartMX xD(T) (0xXX42)	0	0	0	0	0	x^2	x^2	x^2	0	1	0	0	0	0	1	0

1 All RFU bits shall be set to ‘0’ according to the ISO/IEC 14443A-3.

2 For the MIFARE® ProX, and SmartMX Dual & Triple ICs, any bit combinations in the proprietary field are possible.

ISO/IEC 14443A-3 defined bits

Proprietary coded bits

* Application Note, *mifare® Interface Platform, Type Identification Procedure*, Revision 1.3, Page 7, Philips Semiconductor, November 2004.

Select Acknowledge (SAK)

All MIFARE® chips respond to the ISO/IEC 14443A-3 ‘SELECT’ (Select Command, Type A) with the appropriate SAK (Select Acknowledge, Type A). The SAK is one byte long and contains the information as mapped in the table below:

SAK Table*

	SAK							
Bit number	8	7	6	5	4	3	2	1
SAK bit values defined in the ISO/IEC 14443A-3								
Cascade bit set: UID not complete						1[1]		
UID complete, PICC compliant with ISO/IEC 14443-4			1			0		
UID complete, PICC not compliant with ISO/IEC 14443-4			0			0		
SAK response values of different MIFARE® chips with respect to the ISO/IEC 14443A-3								
MIFARE® ultralight (0x04) -- cascade level 1	0	0	0	0	0	1	0	0
MIFARE® ultralight (0x00) -- cascade level 2	0	0	0	0	0	0	0	0
MIFARE® 1K (0x08)	0	0	0	0	1	0	0	0
MIFARE® 4K (0x18)	0	0	0	1	1	0	0	0
MIFARE® DESFire (0x24) -- cascade level 1	0	0	1	0	0	1	0	0
MIFARE® DESFire (0x20) -- cascade level 2	0	0	1	0	0	0	0	0
MIFARE® Pro (0x20)[2]	0	0	1	0[2]	0[2]	0	0	0
MIFARE® Pro (0x08)	0	0	0	0[2]		0	0	0
MIFARE® Pro (0x28)	0	0	1	0		0	0	0
MIFARE® ProX (0x00)[2]	0	0	0	0[2]	0[2]	x[3]	0	0
MIFARE® ProX (0x20)[2]	0	0	1	0[2]	0[2]	x[3]	0	0
MIFARE® ProX (0x08)	0	0	0	0	1	x[3]	0	0
MIFARE® ProX (0x28)	0	0	1	0	1	x[3]	0	0
MIFARE® ProX (0x18)	0	0	0	1	1	x[3]	0	0
MIFARE® ProX (0x38)	0	0	1	1	1	x[3]	0	0
SmartMX xD(T) (0x00)[2]	0	0	0	0[2]	0[2]	x[3]	0	0
SmartMX xD(T) (0x20)[2]	0	0	1	0[2]	0[2]	x[3]	0	0
SmartMX xD(T) (0x08)	0	0	0	0	1	x[3]	0	0
SmartMX xD(T) (0x28)	0	0	1	0	1	x[3]	0	0
SmartMX xD(T) (0x18)	0	0	0	1	1	x[3]	0	0
SmartMX xD(T) (0x38)	0	0	1	1	1	x[3]	0	0

[1] A set cascade bit within the SAK indicates that the UID is not received completely yet and that another anticollision and select loop using the next higher cascade level has to be executed.

[2] Bits 4 and 5 set to ‘0’ in the SAK of the MIFARE® ProX or SmartMX means that the card does not support the MIFARE® Classic protocol.

[3] Depending on ordered configuration and if applicable on the cascade level.

ISO/IEC 14443A-3 defined bits

Proprietary coded bits

* Application Note, *mifare® Interface Platform, Type Identification Procedure*, Revision 1.3, Page 8, Philips Semiconductor, November 2004.

Glossary

AFI	Application family identifier upper 4 bits for family 0 proprietary 1 transportation — mass transit, bus, airlines 2 financial — banking, retail, electronic purse 3 identification — access control 4 telecommunication — telephony 5 medical 6 multimedia — internet services 7 gaming 8 data storage — portable file lower 4 bits for sub family
AID	Application identifier
APDU	Application protocol data unit
ATQ	Answer to request buffer
ATS	Answer to select
CID	Temporary card numbers ranging from 0 thru 14 that allow addressing simultaneously several active ISO 14443-4 cards with a single reader
DRI	Divisor receive integer from the reader to the card
DSI	Divisor send integer from the card to the reader
FSCI	Frame size card integer; max size of the frame accepted by the card default value 2 = 32 bytes
GBP	Gemplus block protocol
MAC	Plain data transfer with DES/TDES cryptographic checksum
NAD	Node address
OROS	Open reader operating system
PUPI	Pseudo unique PICC identifier, 32 bit serial number defined by the customer during personalization
ROS	Reader operating system
SAK	Select acknowledge byte
SNR	Card’s unique ID

6 UHF Functions

Introduction

This section contains information for software developers intending to write applications for UHF-compliant contactless smart cards using Zebra card printer's internal smart card readers.

The Application Programming Interface (API) provides functions to access the internal smart card features.

Required Skills

- Experience in developing applications for the Microsoft Windows environment
- Experience in developing applications using dynamic link libraries (dll)
- Experience with UHF-compliant smart cards

Zebra Card Printers

- P630i
- P640i

Communication Ports

- USB 1.1
- Ethernet

SDK Elements

- ZBRUHFReader.dll
  - 32 bit dynamic link library
  - the dll should be placed in the system directory or the applications directory
- ZBRUHFReader.h
- C++ sample code

Installation

Directory Structure

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

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

System Directories

SDK dll files should be placed in the system directory.

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

Function List

ZBRUHFGetSDKVer 242
ZBRGetHandle 243
ZBRCloseHandle 244
ZBRUHFStartCard 245
ZBRUHFEndCard 246
ZBRUHFEndCardEx 247
ZBRUHFGetReaderVersions 248
ZBRUHFGetCurrentRegion 249
ZBRUHFGetAvailableRegions 250
ZBRUHFSend 251
ZBRUHFReceive 252
ZBRUHFSendReceive 253
ZBRUHFReadTagID 254
ZBRUHFAppendChecksum 255
ZBRUHFWriteTagID 256
ZBRUHFLockTag 257
ZBRUHFWriteTagPasswords 258
ZBRUHFWriteTagData 259
ZBRUHFReadTagData 260
ZBRUHFUnlockTag 261
ZBRUHFReset 262

ZBRUHFGetSDKVer

Description: Gets the SDK version numbers, including major, minor, and engineering versions.

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

Parameters:

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

ZBRGetHandle

Description: Gets a handle for a printer driver.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRCloseHandle

Description: Closes a printer driver handle.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRUHFStartCard

Description: Initializes the UHF reader, configures it for Gen2 protocol, and moves the Gen2 card under the antenna.

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

Parameters:

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

Note: Call this function before sending commands to the reader.

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRUHFEndCard

Description: Inactivates the reader and ejects the Gen2 card.

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

Parameters:

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

Note: Call this function before you call ZBRCloseHandle.

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRUHFEndCardEx

Description: Inactivates the reader and determines whether to eject the card or not.

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

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
eject	[in] eject the card or not: 1 = Yes 0 = No
err	[out]pointer to returned error code

Note: Call this function before you call ZBRCloseHandle.

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRUHFGetReaderVersions

Description: Gets the Boot Loader, Hardware, and Application version numbers.

Syntax:
```
int ZBRUHFGetReaderVersions(
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            *bootLoaderVer,
                        int             *bootLoaderVerSize,
                        byte            *hardwareVer,
                        int             *hardwareVerSize,
                        byte            *fwDate,
                        int             *fwDateSize,
                        byte            *fwVer,
                        int             *fwVerSize
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
bootLoaderVer	[out]boot loader version buffer(byte array)
bootLoaderVerSize	[in, out]: [in] bootLoaderVer buffer size (min 4) [out]number of bytes returned
hardwareVer	hardware version buffer (byte array)
hardwareVerSize	[in, out]: [in] hardwareVer buffer size (min 4) [out]number of bytes returned
fwDate	[out]firmware date buffer (byte array)
fwDateSize	[in, out]: [in] fwDate buffer size (min 4) [out]number of bytes returned
fwVer	[out]firmware version buffer (byte array)
fwVerSize	[in, out]: [in] fwVer buffer size (min 4) [out]number of bytes returned
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

Boot Loader Version:

Product Number (1 byte)	Major release (1 byte)	Minor release (1 byte)	Build release (1 byte)

Firmware Date:

Year (2 bytes)	Month (1 byte)	Day (1 byte)

ZBRUHFGetCurrentRegion

Description: Gets the current region (geographical).

Syntax:
```
int ZBRUHFGetCurrentRegion(
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            *region,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle	
printerType	[in] printer type value, see Appendix B	
region	[out]pointer to returned region code (byte):	
	0x01(NA)	
	0x02(EU)	
	0x03(KR)	
	0x07(EU2)	
err	[out]pointer to returned error code	

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRUHFGetAvailableRegions

Description: Gets a list of supported regions (geographical).

Syntax:
```
int ZBRUHFGetAvailableRegions(
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            *regionList,
                        int             *regionCount,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
regionsList	[out]pointer to available region list (byte array): 0x01(NA) 0x02(EU) 0x03(KR) 0x07(EU2)
regionCount	[out]pointer to number of regions
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRUHFSend

Description: Sends data to the UHF reader.

Syntax:
```
int ZBRUHFSend(
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            *dataIn,
                        int             byteCount,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
dataIn	[in] pointer to send buffer (byte array)
byteCount	[in] number of bytes to send
err	[out]pointer to returned error code

Note: This function can be useful when the user wants to communicate directly with the UHF reader.

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRUHFReceive

Description: Gets data from the UHF reader's buffer.

Syntax:
```
int ZBRUHFReceive(
                    HANDLE          hPrinter,
                    int             printerType,
                    byte            *dataOut,
                    int             dataOutSize,
                    int             *bytesRead,
                    int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
dataOut	[out]pointer to receive buffer (byte array)
dataOutSize	[in] receive buffer size
bytesRead	[out]pointer to number of bytes received
err	[out]pointer to returned error code

Note: This function can be useful when the user wants to communicate directly with the UHF reader.

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRUHFSendReceive

Description: Sends data to the UHF reader and gets data from the UHF reader's buffer.

Syntax:
```
int ZBRUHFSendReceive(
                        HANDLE          hPrinter,
                        int             printerType,
                        unsigned char   *dataIn,
                        int             dataInSize,
                        unsigned char   *dataOut,
                        int             *dataOutSizeNeeded,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
dataIn	[in] pointer to input buffer
dataInSize	[in] size of input buffer
dataOut	[out]pointer to output buffer
dataOutSizeNeeded	[out]pointer to size of output data buffer if successful, size of buffer required if failed
err	[out] pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRUHFReadTagID

Description: Gets the ID that is being used for the current tag protocol.

Syntax:
```
int ZBRUHFReadTagID(
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            *tagID,
                        int             *tagIDSize,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
tagID	[out]pointer to returned tag ID (byte array)
tagIDSize	[out]pointer to returned tag ID size
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRUHFAppendChecksum

Description: Appends a checksum to data in buffer.

Syntax:
```
int ZBRUHFAppendChecksum(
                        unsigned char   *dataIn,
                        int             dataInSize,
                        unsigned char   *dataOut,
                        int             *dataOutSizeNeeded)
```

Parameters:

dataIn	[in] pointer to buffer with data to be appended
dataInSize	[in] size of dataIn buffer
dataOut	[out]pointer to buffer which holds data with checksum appended
dataOutSizeNeeded	[out]pointer to size of dataOut buffer if successful, size of buffer required if failed

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRUHFWriteTagID

Description: Writes data to the ID section of the selected tag.

Syntax:
```
int ZBRUHFWriteTagID(
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            *tagID,
                        int             tagIDSize,
                        int             *err)
```

Parameters:		
	hPrinter	[in] printer driver handle
	printerType	[in] printer type value, see Appendix B
	tagID	[in] pointer tag ID buffer (byte array)
	tagIDSize	[in] tagID buffer size
	err	[out]pointer to returned error code

Return Value:		
	TRUE	successful
	FALSE	failed, check error codes

Error Codes: Appendix A

ZBRUHFLockTag

Description: Locks a specific tag section.

Syntax:
```
int ZBRUHFLockTag(
                    HANDLE          hPrinter,
                    int             printerType,
                    byte            *accessPassword,
                    int             accessPasswordSize,
                    int             memBankBits,
                    int             permBits,
                    int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
accessPassword	[in] pointer to password buffer
accessPasswordSize	[in] password size (min 4 bytes)
memBankBits	[in] banks to lock
permBits	[in] permanent bits
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

Note: memBankBits and permBits:

```
0x01 = User
0x02 = TID
0x04 = EPC
0x08 = Access Password
0x10 = Kill Password
```

ZBRUHFWriteTagPasswords

Description: Writes the tag ID, kill, and access passwords.

Syntax:
```
int ZBRUHFWriteTagPasswords(
                          HANDLE          hPrinter,
                          int             printerType,
                          byte            chipType,
                          byte            option,
                          byte            *tagID,
                          int             tagIDSize,
                          byte            *killPassword,
                          int             killPasswordSize,
                          byte            *accessPassword,
                          int             accessPasswordSize,
                          int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
chipType	[in] chip type, supported type: 0x01
option	[in] option, supported option: 0x01
tagID	[in] pointer to tag ID to write (byte array)
tagIDSize	[in] tag ID size (min 12 bytes)
killPassword	[in] pointer to kill password to write (byte array)
killPasswordSize	[in] kill password size (min 4 bytes)
accessPassword	[in] pointer to access password to write (byte array)
accessPasswordSize	[in] access password size (min 4 bytes)
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRUHFWriteTagData

Description: Writes to data section of a tag.

Syntax:
```
int ZBRUHFWriteTagData(
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            memBank,
                        int             addr,
                        byte            *dataIn,
                        byte            byteCount,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
memBank	[in] memory bank: 0x00(Reserved) 0x01(EPC) 0x02(TID) 0x03(User)
addr	[in] offset from memBank origin
dataIn	[in] pointer to write buffer (byte array)
byteCount	[in] number of bytes to write
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Notes: dataInSize[in] must be an even number of bytes (2, 4, 6); otherwise, the reader will return an error.

Error Codes: Appendix A

ZBRUHFReadTagData

Description: Reads data from the tag.

Syntax:
```
int ZBRUHFReadTagData(
                        HANDLE          hPrinter,
                        int             printerType,
                        byte            memBank,
                        int             addr,
                        byte            wordCount,
                        byte            *dataOut,
                        int             dataOutSize,
                        int             *bytesRead,
                        int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
memBank	[in] memory bank: 0x00(Reserved) 0x01(EPC) 0x02(TID) 0x03(User)
addr	[in] offset from memBank origin
wordCount	[in] number of words to read
dataOut	[out]pointer to data buffer
dataOutSize	[in] dataOut buffer size
bytesRead	[out]pointer to number of bytes read
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRUHFUnlockTag

Description: Unlocks a specific tag section.

Syntax:
```
int ZBRUHFUnlockTag(
                          HANDLE          hPrinter,
                          int             printerType,
                          unsigned char   *accessPassword,
                          int             accessPasswordSize,
                          int             memBankBits,
                          int             permBits,
                          int             *err)
```

Parameters:

hPrinter	[in] printer driver handle
printerType	[in] printer type value, see Appendix B
accessPassword	[in] pointer to buffer with password
accessPasswordSize	[in] size of password buffer
memBankBits	[in] banks to unlock
permBits	[in] permanent bits
err	[out]pointer to returned error code

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

Note: memBankBits and permBits

```
0x01 = User
0x02 = TID
0x04 = EPC
0x08 = Access Password
0x10 = Kill Password
```

ZBRUHFReset

Description: Resets a printer.

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

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

7 Programming Examples

The programming examples in this section show how to use Zebra SDK functions and Windows API to perform printer-specific operations:

Basic Card Printing and Magnetic Stripe Encoding 264
Contact Smart Card 267
MIFARE 270
UHF 275
Barcode 277

Basic Card Printing and Magnetic Stripe Encoding

The following example shows how to encode three tracks of magnetic stripe data, and then print an image and text.

```
// C++ ZBRPrinter.dll and ZBRGraphics.dll Example
//*****************************************************************************************
//*****************************************************************************************

#include "windows.h”
#include "stdafx.h”

// Type Defines for ZBRPrinter.dll functions
// -----------------------------------------------------------------------------------------
// -----------------------------------------------------------------------------------------

       // Handle functions

typedef int (CALLBACK *funcGetHandle)(HANDLE *prnHandle, char *devName, int *prnType,
                                      int* errValue);
funcGetHandle getHandle;

typedef int (CALLBACK *funcCloseHandle)(HANDLE prnHandle, int *errValue);
funcCloseHandle closeHandle;

       // Position functions

typedef int (CALLBACK *funcMovePrintReady)(HANDLE prnHandle, int prnType, int *errValue);
funcMovePrintReady movePrintReady;

       // Magnetic encoder functions

typedef int (CALLBACK *funcReadMag)(HANDLE prnHandle, int prnType, int tracks,
                                      char *trkBuf1, int *sz1, char *trkBuf2, int *sz2,
                                      char *trkBuf3, int *sz3, int *errValue);
funcReadMag readMag;

typedef int (CALLBACK *funcWriteMag)(HANDLE prnHandle, int prnType, int tracks,
                                      char *trkBuf1, char *trkBuf2, char *trkBuf3,
                                      int *errValue);
funcWriteMag writeMag;

// Type Defines for ZBRGraphics.dll functions
// -----------------------------------------------------------------------------------------
// -----------------------------------------------------------------------------------------

       // Graphic buffer functions

typedef int (CALLBACK *funcInitGraphics)(char *devName, HDC *hDC, int *errValue);
funcInitGraphics initGraphics;

typedef int (CALLBACK *funcPrintGraphics)(HDC hDC, int *errValue);
funcPrintGraphics printGraphics;

typedef int (CALLBACK *funcCloseGraphics)(HDC hDC, int *errValue);
funcCloseGraphics closeGraphics;



// Continued on next page
```

```
       // Draw functions

typedef int (CALLBACK *funcDrawImageRect)(char *filename, int x, int y, int width,
                                          int height, int *errValue);
funcDrawImageRect drawImageRect;

typedef int (CALLBACK *funcDrawText)(int x, int y, char *txt, char *fnt, int fntSize,
                                          int fntStyle, int color, int *errValue);
funcDrawText drawText;

// Main
// ------------------------------------------------------------------------------------
// ------------------------------------------------------------------------------------

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine,
                     int nCmdShow)
{
       HINSTANCE       dll;
       HANDLE          prnHandle;
       int             errValue,
                       prnType,
                       ret;

       // ZBRPrinter.dll functions
       // ------------------------------------------------------------------------------

       dll = LoadLibrary("ZBRPrinter.dll”);
       if (dll == NULL) return 1;

              // Printer functions

       closeHandle     = (funcCloseHandle)GetProcAddress(dll, "ZBRCloseHandle”);
       getHandle       = (funcGetHandle)GetProcAddress(dll, "ZBRGetHandle”);
       movePrintReady  = (funcMovePrintReady)GetProcAddress(dll, "ZBRPRNMovePrintReady”);

              // Magnetic encoder functions

       readMag         = (funcReadMag)GetProcAddress(dll, "ZBRPRNReadMag”);
       writeMag        = (funcWriteMag)GetProcAddress(dll, "ZBRPRNWriteMag”);

              // Get a handle to the printer driver

       ret = getHandle(&prnHandle, "Zebra P330i USB Card Printer”, &prnType, &errValue);

       int tracks = 7;           // 7 write or reads all three tracks
                                 // 1 write or read track 1 only
                                 // 2 write or read track 2 only
                                 // 4 write or read track 3 only
                                 // or these values to write or read multiple tracks

              // Write to all three magnetic stripe tracks

       ret = writeMag(prnHandle,
                                 prnType,
                                 tracks,
                                 "B5018780618000015 41^John Doe ^4912101”,    // track 1
                                 "5018780618000015 41=4912101678”,            // track 2
                                 "00000000000000000 1241”,                    // track 3
                                 &errValue);



// Continued on next page
```

```
        // Variables for reading the magnetic stripe

    char   trkBuf1[255], trkBuf2[255], trkBuf3[255]; // buffer to receive track data
    int    sz1, sz2, sz3;                            // returned byte count in buffers

    for (int i=0; i < sizeof(trkBuf1); i++) {
        trkBuf1[i] = trkBuf2[i] = trkBuf3[i] = 0;
    }
        // Read all three magnetic stripe tracks

    ret = readMag(prnHandle, prnType, tracks, trkBuf1, &sz1, trkBuf2, &sz2, trkBuf3,
                &sz3, &errValue);

        // Move the card to the printing location

    ret = movePrintReady(prnHandle, prnType, &errValue);

        // Close the printer driver handle

    ret = closeHandle(prnHandle, &errValue);

    dll = NULL;

    // ZBRGraphics.dll functions
    // ------------------------------------------------------------------------------

    dll = LoadLibrary("ZBRGraphics.dll”);
    if( dll == NULL ) return 2;

    closeGraphics   = (funcCloseGraphics)GetProcAddress(dll, "ZBRGDICloseGraphics”);
    drawImageRect   = (funcDrawImageRect)GetProcAddress(dll, "ZBRGDIDrawImageRect”);
    drawText        = (funcDrawText)GetProcAddress(dll, "ZBRGDIDrawText”);
    initGraphics    = (funcInitGraphics)GetProcAddress(dll, "ZBRGDIInitGraphics”);
    printGraphics   = (funcPrintGraphics)GetProcAddress(dll, "ZBRGDIPrintGraphics”);

    HDC hDC = NULL;

        // Initialize the graphics buffer

    ret = initGraphics("Zebra P330i USB Card Printer”, &hDC, &errValue);

        // Draw in the graphic buffer an image and text

    ret = drawImageRect("Zebra.bmp”, 50, 50, 200, 150, &errValue);
    ret = drawText(250, 250, "Text Here”, "Arial”, 12, 0x01, 0x808080, &errValue);

        // Print the image in the graphics buffer

    ret = printGraphics(hDC, &errValue);

        // Close the graphics buffer

    ret = closeGraphics(hDC, &errValue);

    return 0;
}
```

Contact Smart Card

The following example demonstrates how to write to and read from a SLE 4442 smartcard:

```
// C++ ZBRGC.dll (GemCore) Example
// ********************************************************************************
// ********************************************************************************

#include "windows.h"
#include "stdafx.h"

#define ZBR_SYNCHRONOUS 1
#define ZBR_ISO_78163 2

// Type Defines for ZBRGC.dll functions
// ------------------------------------------------------------------------
// ------------------------------------------------------------------------

       // Handle functions

typedef int (CALLBACK *funcGetHandle)(HANDLE *prnHandle, char *devName, int *prnType,
                                      int *errValue);
funcGetHandle getHandle;

typedef int (CALLBACK *funcCloseHandle)(HANDLE prnHandle, int *errValue);
funcCloseHandle closeHandle;

       // Card functions

typedef int (CALLBACK *funcEndCardEx)(HANDLE prnHandle, int prnType, int eject,
                                      int *errValue);
funcEndCardEx endCardEx;

typedef int (CALLBACK *funcExchangeData)(HANDLE prnHandle, int prnType,
                                      bytebyte *dataIn, int dataInSize,
                                      bytebyte *dataOut, int dataOutSize,
                                      int *respSize, int *errValue);
funcExchangeData exchangeData;

typedef int (CALLBACK *funcSetCardType)(HANDLE prnHandle, int prnType, int cardType,
                                      int *errValue);
funcSetCardType setCardType;

typedef int (CALLBACK *funcStartCard)(HANDLE prnHandle, int prnType, int *errValue);
funcStartCard startCard;

// Main
// ------------------------------------------------------------------------------------
// ------------------------------------------------------------------------------------

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int
nCmdShow)
{
       HINSTANCE       dll;
       HANDLE          prnHandle;
       int             errValue,
                       prnType,
                       respSize,
                       ret;
       bytebyte        dataOut[1024];

// Continued on next page
```

```
        // Main Functions
        // ----------------------------------------------------------------------------
        // ----------------------------------------------------------------------------

        dll = LoadLibrary("ZBRGC.dll");
        if (dll == NULL) return 1;

               // Get a handle to the printer driver

        getHandle = (funcGetHandle)GetProcAddress(dll, "ZBRGetHandle");
        ret = getHandle(&prnHandle, "Zebra P330i USB Card Printer", &prnType, &errValue);

               // Position card for encoding

        startCard = (funcStartCard)GetProcAddress(dll, "ZBRGCStartCard");
        ret = startCard(prnHandle, prnType, &errValue);

               // Set card type synchronous card

        setCardType = (funcSetCardType)GetProcAddress(dll, "ZBRGCSetCardType");
        ret = setCardType(prnHandle, prnType, ZBR_SYNCHRONOUS, &errValue);

               // Reset a SLE4442 card

        byte rstData[] = {0x16, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0x11,
                                    0x12};
        exchangeData = (funcExchangeData)GetProcAddress(dll, "ZBRGCExchangeData");
        ret = exchangeData(prnHandle, prnType, rstData, (int)sizeof(rstData), dataOut,
                                (int)sizeof(dataOut), &respSize, &errValue);

               // Authentication for a SLE4442 card

        byte            chkCode[]  = {  0x16,0x00,0x20,0x00,0x00,0x03,0xFF,0xFF,
                                        0xFF,0x00,0x63,0xBE,0x00,0x1F,0xBF,0x00,
                                        0x1C,0xBA,0x03,0x19,0x21,0x51,0x71,0x41,
                                        0x74,0x31,0x93,0x74,0xFF,0x93,0x93,0x41,
                                        0x71,0x53,0xA2,0x7D,0x08,0x13,0x40,0x08,
                                        0xDD,0xFB,0x62,0x98,0x40,0x62,0x6D,0x00,
                                        0xC3,0x13,0xDD,0xFD,0xFD,0x71,0x41,0x74,
                                        0x39,0x93,0xE4,0x93,0xED,0x93,0x41,0x71,
                                        0x51,0x61,0x71,0xE1,0x50,0xFB,0x0F,0x41,
                                        0x74,0x33,0x93,0xEF,0x93,0xE7,0x09,0x93,
                                        0x41,0x71,0x51,0x61,0x71,0xE1,0x50,0xFB,
                                        0xDA,0xEC,0x7D,0x07,0x41,0x74,0x39,0x93,
                                        0xE4,0x93,0xED,0x93,0x41,0x71,0x51,0x61,
                                        0x71,0xE1,0x50,0xFB,0x61,0x42};

        ret = exchangeData(prnHandle, prnType, chkCode, (int)sizeof(chkCode), dataOut,
                                (int)sizeof(dataOut), &respSize, &errValue);

               // Write data to a SLE4442 card

        int             dataSize   = 10;
        byte            addr       = 32;
        byte            wrCode[]   = {  0x00,0x2C,0x21,0x51,0x71,0x41,0xBE,0x00,
                                        0x04,0x74,0x38,0x80,0x0C,0xBE,0x80,0x04,
                                        0x74,0x3C,0x80,0x05,0xBE,0xC0,0x14,0x74,
                                        0x39,0x93,0xEF,0x93,0xE7,0x09,0x93,0x41,
                                        0x71,0x51,0x61,0x71,0xE1,0x50,0xFB,0x0F,
                                        0xDA,0xD9,0x42,0x62,0x6D,0x00};
        byte            dataIn[10];
        for (int i=0; i<dataSize; i++) dataIn[i] = 0x41 + i;

// Continued on next page
```

```
        byte wrCmd[62];
        wrCmd[0] = 0x16;
        wrCmd[1] = 0x00;
        wrCmd[2] = 0xD0;
        wrCmd[3] = 0x00;
        wrCmd[4] = addr;
        wrCmd[5] = dataSize;

        for(int i=6; i < 6+dataSize; i++)
               wrCmd[i] = dataIn[i-6];

        for(int i = 6 + dataSize; i < 6 + dataSize + (int)sizeof(wrCode); i++)
               wrCmd[i] = wrCode[i-6-dataSize];

        ret = exchangeData(prnHandle, prnType, wrCmd, (int)sizeof(wrCmd), dataOut,
                                (int)sizeof(dataOut), &respSize, &errValue);

               // Read data from a SLE4442

        byte            rdCode[]   = {  0x16,0x00,0xB0,0x00,addr,0x00,
                                        (char)dataSize,0x3E,0x21,0xBE,0x00,0x14,
                                        0x51,0x71,0x41,0x74,0x30,0x93,0xEF,0x93,
                                        0x74,0xFF,0x93,0x41,0x71,0x53,0xF6,0x08,
                                        0xDB,0xFB,0xA2,0x42,0xBE,0x80,0x02,0x80,
                                        0x03,0xBE,0xC0,0x1B,0x51,0x71,0x41,0xBE,
                                        0xC0,0x04,0x74,0x31,0x80,0x02,0x74,0x34,
                                        0x93,0x74,0xFF,0x93,0x93,0x41,0x71,0x7B,
                                        0x04,0x53,0xF6,0x08,0xDB,0xFB,0x42,0x62,
                                        0x6D,0x00};

        ret = exchangeData(prnHandle, prnType, rdCode, (int)sizeof(rdCode), dataOut,
                                (int)sizeof(dataOut), &respSize, &errValue);

               // End process and eject card

        endCardEx = (funcEndCardEx)GetProcAddress(dll, "ZBRGCEndCardEx");
        ret = endCardEx(prnHandle, prnType, 1, &errValue);

               // Closes the printer driver handle

        closeHandle = (funcCloseHandle)GetProcAddress(dll, "ZBRCloseHandle");
        ret = closeHandle(prnHandle, &errValue);

        dll = NULL;

        return 0;
}
```

MIFARE

The following example shows how to use the contactless smart card SDK (zbrgpmf.dll) to write to and read from a MIFARE contactless card:

```
// ZBRGPMF_SampleApp.cpp : Defines the entry point for the console application.
// ********************************************************************************************
// ********************************************************************************************

#include "stdio.h”
#include "windows.h”
#include "ZBRGPMFApp.h”
#include "ZBRGPMF.h”

#define ZBR_ERROR_NO_ERROR0
#define MAX_RESPONSE_SIZE511

//
// Load Zebra printer SDK functions
//

BOOL LoadZBRSDKFunctions()
{
        // Load the Zebra MIFARE SDK library
        HMODULE hModule = LoadLibrary("ZBRGPMF.dll”);
        if (hModule)
                printf("The DLL has been successfully loaded.\n”);
        else
        {
                printf("Error loading Zebra SDK DLL.\n”);
                return FALSE;
        }

        // Get the functions
        zsdkGetHandle = (ZBRGetHandle)GetProcAddress(hModule, "ZBRGetHandle”);
        zsdkSetupPrinter = (ZBRSetupPrinter)GetProcAddress(hModule, "ZBRSetupPrinter”);
        zsdkCloseHandle = (ZBRCloseHandle)GetProcAddress(hModule, "ZBRCloseHandle”);
        zsdkStartCard = (ZBRMFStartCard)GetProcAddress(hModule, "ZBRGPMFStartCard”);
        zsdkEndCard = (ZBRMFEndCard)GetProcAddress(hModule, "ZBRGPMFEndCard”);
        zsdkEndCardEx= (ZBRMFEndCardEx)GetProcAddress(hModule, "ZBRGPMFEndCardEx”);
        zsdkGetVersion = (ZBRMFSDKGetVer)GetProcAddress(hModule, "ZBRGPMFSDKGetVer”);

        zsdkGetReaderId = (ZBRMF_Reader_GetID)GetProcAddress(hModule, "ZBRGPMF_Reader_GetID”);
        zsdkGetReaderFW= (ZBRMF_Reader_GetFirmware)GetProcAddress(hModule,
                        "ZBRGPMF_Reader_GetFirmware”);
        zsdkGetModeAndGBPAddress = (ZBRMF_Reader_GetModeAndGBPAddress)GetProcAddress(hModule,
                        "ZBRGPMF_Reader_GetModeAndGBPAddress”);

        zsdkReadModulationType = (ZBRMF_RF_ReadModulationType)GetProcAddress(hModule,
                        "ZBRGPMF_RF_ReadModulationType”);
        zsdkChangeModulationType = (ZBRMF_RF_ChangeModulationType)GetProcAddress(hModule,
                        "ZBRGPMF_RF_ChangeModulationType”);
        zsdkControlRF = (ZBRMF_RF_Control)GetProcAddress(hModule, "ZBRGPMF_RF_Control”);

        zsdkISO14443_B_GetCard = (ZBRMF_ISO14443_3_B_GetCard)GetProcAddress(hModule,
                        "ZBRGPMF_ISO14443_3_B_GetCard”);

        zsdkISO14443_4_A_B_Exchange_TCL = (ZBRMF_ISO14443_4_A_B_Exchange_T_CL)
                        GetProcAddress(hModule, "ZBRGPMF_ISO14443_4_A_B_Exchange_T_CL”);

        zsdkISO14443_3_A_GetCardA_TCL = (ZBRMF_ISO14443_3_A_GetCardA_T_CL)
                        GetProcAddress(hModule, "ZBRGPMF_ISO14443_3_A_GetCardA_T_CL”);
        zsdkISO14443_3_A_GetCard = (ZBRMF_ISO14443_3_A_GetCard)GetProcAddress(hModule,
                        "ZBRGPMF_ISO14443_3_A_GetCard”);
        zsdkISO14443_3_A_RequestA = (ZBRMF_ISO14443_3_A_RequestA)GetProcAddress(hModule,
                        "ZBRGPMF_ISO14443_3_A_RequestA”);

// Continued on next page
```

```
        zsdkISO14443_3_A_Anticollision = (ZBRMF_ISO14443_3_A_Anticollision)
                        GetProcAddress(hModule, "ZBRGPMF_ISO14443_3_A_Anticollision”);
        zsdkISO14443_3_A_Select = (ZBRMF_ISO14443_3_A_Select)GetProcAddress(hModule,
                        "ZBRGPMF_ISO14443_3_A_Select”);
        zsdkISO14443_3_A_Halt = (ZBRMF_ISO14443_3_A_Halt)GetProcAddress(hModule,
                        "ZBRGPMF_ISO14443_3_A_Halt”);

        zsdkMF_LoadKey = (ZBRMF_LoadKey)GetProcAddress(hModule, "ZBRGPMF_LoadKey”);
        zsdkMF_Authenticate = (ZBRMF_Authenticate)GetProcAddress(hModule,
                        "ZBRGPMF_Authenticate”);
        zsdkMF_Write = (ZBRMF_Write)GetProcAddress(hModule, "ZBRGPMF_Write”);
        zsdkMF_Read = (ZBRMF_Read)GetProcAddress(hModule, "ZBRGPMF_Read”);
        zsdkMF_Transfer = (ZBRMF_Transfer)GetProcAddress(hModule, "ZBRGPMF_Transfer”);
        zsdkMF_AddValue = (ZBRMF_AddValue)GetProcAddress(hModule, "ZBRGPMF_AddValue”);
        zsdkMF_SubtractValue = (ZBRMF_SubtractValue)GetProcAddress(hModule,
                        "ZBRGPMF_SubtractValue”);
        zsdkMF_Restore = (ZBRMF_Restore)GetProcAddress(hModule, "ZBRGPMF_Restore”);

        zsdkMF_C_Write = (ZBRMF_C_Write)GetProcAddress(hModule, "ZBRGPMF_C_Write”);
        zsdkMF_C_Read = (ZBRMF_C_Read)GetProcAddress(hModule, "ZBRGPMF_C_Read”);
        zsdkMF_C_SetAccessConditions = (ZBRMF_C_SetAccessConditions)
                        GetProcAddress(hModule, "ZBRGPMF_C_SetAccessConditions”);
        zsdkMF_C_CreateValueBlock = (ZBRMF_C_CreateValueBlock)GetProcAddress(hModule,
                        "ZBRGPMF_C_CreateValueBlock”);
        zsdkMF_C_ReadValue = (ZBRMF_C_ReadValue)GetProcAddress(hModule,
                        "ZBRGPMF_C_ReadValue”);
        zsdkMF_C_SubtractValue = (ZBRMF_C_SubtractValue)GetProcAddress(hModule,
                        "ZBRGPMF_C_SubtractValue”);
        zsdkMF_C_AddValue = (ZBRMF_C_AddValue)GetProcAddress(hModule,
                        "ZBRGPMF_C_AddValue”);

        zsdkMF_B_CreatePurse = (ZBRMF_B_CreatePurse)GetProcAddress(hModule,
                        "ZBRGPMF_B_CreatePurse”);
        zsdkMF_B_DebitPurse = (ZBRMF_B_DebitPurse)GetProcAddress(hModule,
                        "ZBRGPMF_B_DebitPurse”);
        zsdkMF_B_CreditPurse = (ZBRMF_B_CreditPurse)GetProcAddress(hModule,
                        "ZBRGPMF_B_CreditPurse”);
        zsdkMF_B_ReadPurse = (ZBRMF_B_ReadPurse)GetProcAddress(hModule,
                        "ZBRGPMF_B_ReadPurse”);

        zsdkMF_ExchangeData = (ZBRMF_ExchangeData)GetProcAddress(hModule,
                        "ZBRGPMF_TransparentExchange”);
        zsdkMF_ExchangeTimeout = (ZBRMF_ExchangeTimeout)GetProcAddress(hModule,
                        "ZBRGPMF_TransparentExchangeTimeout”);

        return true;
}

int main(int argc, char* argv[])
{
        HANDLE hPrinter = NULL;
        int printerType = 0;
        int err = 0;
        int error = 0;

        LoadZBRSDKFunctions();

        // Get Handle To Printer
        zsdkGetHandle(&hPrinter, "Zebra P330i USB Card Printer”, &printerType, &error);
        if (error != ZBR_ERROR_NO_ERROR)
        {
                printf("\nGet Printer Handle Error: %d”, error);
                return 0;
        }



// Continued on next page
```

```
        // Start Card
        err = zsdkStartCard(hPrinter, printerType, &error);
        if (!err || error != ZBR_ERROR_NO_ERROR)
        {
               printf("\nStart Card Error: %d", error);
               return 0;
        }

        // Turn RF Field On
        byte RF_On = 1, RF_Off = 2, RF_Reset = 3;
        err = zsdkControlRF(hPrinter, printerType, RF_On, &error);
        if (!err || error != ZBR_ERROR_NO_ERROR)
        {
               printf("\nRF Error: %d", error);
               return 0;
        }

        // Timeout and Card Management Struct Definitions
        sCARD_AND_TIMEOUT cardAndTimeout;           // Card And Timeout Structure
        cardAndTimeout.ucIsTimeoutSpecified = 0x00; // Timeout specified
        cardAndTimeout.ucTimeout_50msBased = 0x28;  // Timeout value
        cardAndTimeout.ucCard = 0x00;               // First Card

        // Card Target is the first ISO14443A Card Found in Field
        sGET_CARD_A getCardA;                       // Card A Info Structure
        byte serialNumberA[10];                     // Three cascade level length
        getCardA.pucSerialNumber = serialNumberA;
        getCardA.iSerialNumberSize = sizeof(serialNumberA);

        // Get ISO14443 A Card
               /* Performs the following sequence of commands
                        - RF Reset
                        - Request A
                        - Anticollision
                        - Select
               */
        err = zsdkISO14443_3_A_GetCard(hPrinter, printerType, &cardAndTimeout, &getCardA,
               &error);
        if (!err || error != ZBR_ERROR_NO_ERROR)
        {
               printf("\n\nISO14443_3_A_GetCard Error: %d\n\n", error);
               return 0;
        }


        byte cardType = 0x00;                       // 0x00 = GEMEASY_8000/MIFARE 1K
                                                    // 0x02 = GEMCOMBI/MIFARE 4K with
                                                    //  automatic block value
                                                    // 0x03 = GEMEASY_32000/MIFARE 4K
        // Determine Card Type
        if (((getCardA.ucSAK & 0x08) && !(getCardA.ucSAK & 0x10))
               && getCardA.usATQA == 0x04)
        {
               cardType = 0x00;
               printf("\n\nMIFARE 1K");
        }
        else if ( ((getCardA.ucSAK & 0x08) && (getCardA.ucSAK & 0x10))
               && getCardA.usATQA == 0x02 )
        {
               cardType = 0x03;
               printf("\n\nMIFARE 4K");
        }

// Continued on next page
```

```
else if ( (getCardA.ucSAK & 0x20) && getCardA.usATQA == 0x0344 )
{
       printf("\n\nDESFIRE Card Detected");

       printf("\n\nCard Serial Number: ");
       for (int i = 0; i <= getCardA.iSerialNumberSize - 1; i++)
              printf("%02x", getCardA.pucSerialNumber[i]);

       printf("\nSelect Acknowledge: %02x\n", getCardA.ucSAK);  // Select Ack Type A
       printf("Answer To Request: %02x\n", getCardA.usATQA);    // Ans To Req Type A

       err = zsdkEndCard(hPrinter, printerType, &error);
       return 0;
}
else
{
       printf("\n\nUnknown Card Type, Not a MIFARE card");
       err = zsdkEndCard(hPrinter, printerType, &error);
       return 0;
}

// Print Card Information to Console
printf("\n\nCard Serial Number: ");
for (int i = 0; i <= getCardA.iSerialNumberSize - 1; i++)
       printf("%02x", getCardA.pucSerialNumber[i]);

printf("\nSelect Acknowledge: %02x\n", getCardA.ucSAK);  // Select Ack Type A
printf("Answer To Request: %02x\n", getCardA.usATQA);    // Ans To Req Type A

// Perform Write And Read Operations On MIFARE Cards

// Variable Declarations
byte authentication = 0x01;          // 0 = No Authentication, 1 = KeyA, 2 = KeyB
byte keyAB = 0x00;                   // 0 = Key A, 1 = Key B
byte blockNumber = 0x04;             // Selected Block Number
byte writeVerify = 0x01;             // 0 = No Write Verify, 1 = Write Verify

byte* keyDataA = new byte[6];        // Key to load (6 byte key value)
int dataSize = 48;                   // Write data size
byte* data = new byte[dataSize];     // Write data array (3 blocks)
int outDataSize = 16;                // Read data size (1 block)
byte* dataOut = new byte[outDataSize]; // Read data array

// Create Key Data
keyDataA[0] = 0xFF;
keyDataA[1] = 0xFF;
keyDataA[2] = 0xFF;
keyDataA[3] = 0xFF;
keyDataA[4] = 0xFF;
keyDataA[5] = 0xFF;

// Load 15 Keys To Reader
for (i = 0; i <= 60; i += 4)
{
       err = zsdkMF_LoadKey(hPrinter, printerType, i, keyAB, keyDataA, &error);
       if (!err || error != ZBR_ERROR_NO_ERROR)
       {
              printf("\nLoad Key Error: %d", error);
              return 0;
       }
}


// Continued on next page
```

```
	// Create Data to Write to card - 3 Blocks = 48 bytes
	for (int ii = 0; ii < 48; ii++)
		data[ii] = 0xAA;

	// Write then Read blocks 4-63 skipping sector trailers
	for (ii = 4; ii < 63; ii++)
	{
		// Write Data to 3 Blocks In Sector
		err = zsdkMF_C_Write(hPrinter, printerType, cardType, blockNumber,
			uthentication, writeVerify, data, &dataSize, &error);
		if (!err || error != ZBR_ERROR_NO_ERROR)
		{
			printf("\nWrite Error: %d”, error);
			return 0;
		}
		// Print Written Data to Console Window
		printf("\n\nData Written To Block %d\n”, blockNumber);
		for (int i = 0; i <= dataSize - 1; i++)
			printf("%02x”, data[i]);

		// Read Individual Blocks in each Sector
		for (int x = 4; x < 7; x++)
		{
			// Read Block Data
			err = zsdkMF_Read(hPrinter, printerType, cardType, blockNumber,
				dataOut, &outDataSize, &error);
			if (!err || error != ZBR_ERROR_NO_ERROR)
			{
				printf("\nRead Error: %d”, error);
				return 0;
			}

			// Print Read Data to Console Window
			printf("\nData Read From Block %d:”, blockNumber);
			for (i = 0; i <= outDataSize - 1; i++)
				printf("%02x”, dataOut[i]);

			blockNumber++;
		}
		ii += 3;
		blockNumber++;
	}

	// Turn off RF field
	err = zsdkControlRF(hPrinter, printerType, RF_Off, &error);
	if (!err || error != ZBR_ERROR_NO_ERROR)
	{
		printf("\nRF Error: %d”, error);
		return 0;
	}

	// End Card and Eject
	err = zsdkEndCardEx(hPrinter, printerType, true, &error);
	if (!err || error != ZBR_ERROR_NO_ERROR)
	{
		printf("\nEnd Card Error: %d”, error);
		return 0;
	}

	return 0;
}
```

UHF

The following example demonstrates how to send a command to the UHF encoder and receive its response, as well as how to write data to and read data from a UHF smartcard.

```
// ZBRUHFApp.cpp : Defines the entry point for the console application.
#include “stdafx.h”
#include “stdio.h”
#include “windows.h”
#include “ZBRUHFReader.h”

// SDK functions to be loaded for use in the application
ZBRGetHandle                    getHandle = NULL;
ZBRCloseHandle                  closeHandle = NULL;
ZBRUHFStartCard                 startCard = NULL;
ZBRUHFEndCard                   endCard = NULL;
ZBRUHFGetReaderVersions         getReaderVersions = NULL;
ZBRUHFWriteTagData              writeTagData = NULL;
ZBRUHFReadTagData               readTagData = NULL;

// Loads SDK functions above
// Returns:
// TRUE:  all functions have been successfully loaded
// FALSE: function/library loading has failed

bool loadUHFFunctions(void)
{
       // Load the library
       HMODULE hModule = NULL;
       hModule = LoadLibrary(“ZBRUHFReader.dll”);

       // Can’t load library
       if (!hModule)
              return false;

       // Load functions
       getHandle = (ZBRGetHandle)GetProcAddress(hModule, “ZBRGetHandle”);
       if (!getHandle)
              return false;
       closeHandle = (ZBRCloseHandle)GetProcAddress(hModule, “ZBRCloseHandle”);
       if (!closeHandle)
              return false;
       startCard = (ZBRUHFStartCard)GetProcAddress(hModule, “ZBRUHFStartCard”);
       if (!startCard)
              return false;
       endCard = (ZBRUHFEndCard)GetProcAddress(hModule, “ZBRUHFStartCard”);
       if (!endCard)
              return false;
       getReaderVersions = (ZBRUHFGetReaderVersions)GetProcAddress(hModule,
       “ZBRUHFGetReaderVersions”);
       if (!getReaderVersions)
              return false;
       writeTagData = (ZBRUHFWriteTagData)GetProcAddress(hModule, “ZBRUHFWriteTagData”);
       if (!writeTagData)
              return false;
       readTagData = (ZBRUHFReadTagData)GetProcAddress(hModule, “ZBRUHFReadTagData”);
       if (!readTagData)
              return false;

       // Everything has succeeded, return true
       return true;
}

// Continued on next page
```

```
int main(int argc, char* argv[])
{
	// If unable to load functions, exit
	if (!loadUHFFunctions())
		return 0;

	// Get printer Handle
	HANDLE hPrinter = NULL;
	int printerType = 0;
	int err = 0;
	getHandle(&hPrinter, “Zebra P430i USB Card Printer”, &printerType, &err);

	// Unable to get printer handle, exit
	if (!hPrinter)
		return 0;

	// Load card into UHF encoding station
	startCard(hPrinter, printerType, &err);

	// Card loading failed
	if (err)
		return 0;

	// Write tag data
	// memory bank: 0x01
	byte memBank = 0x01;
	// address  : 0x02
	int addr = 0x02;
	// bytes  : 4 bytes. byte count = 4, word count = 2
	byte byteCount = 4;
	// tag data in hex - “11 22 33 44”
	byte tagData[] = {0x11, 0x22, 0x33, 0x44};

	writeTagData(hPrinter, printerType, memBank, addr, tagData, byteCount, &err);

	// If write tag data filed
	if (err)
		return 0;

	// read tag data
	byte tagDataOut[255];
	// tagDataOut size = 255 bytes
	int tagDataOutSize = 255;
	// word count = 2, 4 bytes
	int wordCount = 2;
	// No bytes read yet, so bytes read is 0
	int bytesRead = 0;

	readTagData(hPrinter, printerType, memBank, addr, wordCount, tagDataOut,
	tagDataOutSize, &bytesRead, &err);

	// If read tag data failed
	if (err)
		return 0;

	// Compare tagData with tagDataOut to see if they are identical
	// memcmp return 0 if yes, other than 0 if not
	if (memcmp(tagData, tagDataOut, wordCount * 2) == 0)
		printf(“Data verified successfully.\n”);
	else
		printf(“Data verification failed.\n”);

	return 0;
}
```

Barcode

The following example demonstrates how to print a barcode on a card:

```
// C++ ZBRGraphics.dll Barcode Example
//*****************************************************************************************
//*****************************************************************************************

#include "windows.h"
#include "stdafx.h"

// Type Defines for ZBRGraphics.dll functions
// -----------------------------------------------------------------------------------------
// -----------------------------------------------------------------------------------------

       // Graphic buffer functions

typedef int (CALLBACK *funcInitGraphics)(char *devName, HDC *hDC, int *errValue);
funcInitGraphics initGraphics;

typedef int (CALLBACK *funcPrintGraphics)(HDC hDC, int *errValue);
funcPrintGraphics printGraphics;

typedef int (CALLBACK *funcCloseGraphics)(HDC hDC, int *errValue);
funcCloseGraphics closeGraphics;

       // Draw functions

typedef int (CALLBACK *funcDrawBarCode)(int x, int y, int rotation, int barcodeType,
                                        int barcodeWidth, int barcodeMultiplier,
                                        int barcodeHeight, int textUnder,
                                        LPSTR barcodeData, int *errValue);
funcDrawBarCode drawBarCode;

typedef int (CALLBACK *funcDrawImageRect)(char *filename, int x, int y, int width,
                                        int height, int *errValue);
funcDrawImageRect drawImageRect;

typedef int (CALLBACK *funcDrawText)(int x, int y, char *txt, char *fnt, int fntSize,
                                        int fntStyle, int color, int *errValue);
funcDrawText drawText;

// Main
// -----------------------------------------------------------------------------------------
// -----------------------------------------------------------------------------------------

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine,
                     int nCmdShow)
{
       HINSTANCE       dll;
       int             errValue,
                       ret;



// Continued on next page
```

```
        // ZBRGraphics.dll functions
        // ----------------------------------------------------------------------------

        dll = LoadLibrary("ZBRGraphics.dll");
        if( dll == NULL ) return 1;

        closeGraphics   = (funcCloseGraphics)GetProcAddress(dll, "ZBRGDICloseGraphics");
        drawBarCode     = (funcDrawBarCode)GetProcAddress(dll, "ZBRGDIDrawBarCode");
        drawImageRect   = (funcDrawImageRect)GetProcAddress(dll, "ZBRGDIDrawImageRect");
        drawText        = (funcDrawText)GetProcAddress(dll, "ZBRGDIDrawText");
        initGraphics    = (funcInitGraphics)GetProcAddress(dll, "ZBRGDIInitGraphics");
        printGraphics   = (funcPrintGraphics)GetProcAddress(dll, "ZBRGDIPrintGraphics");

        HDC hDC = NULL;

               // Initialize the graphics buffer

        ret = initGraphics("Zebra P330i USB Card Printer", &hDC, &errValue);

               // Draw in the graphic and image and text

        ret = drawImageRect("Zebra.bmp", 50, 50, 200, 150, &errValue);
        ret = drawText(250, 250, "Barcode Example", "Arial", 12, 0x01, 0x808080, &errValue);

               // Barcode variables

        int startX                = 280;
        int startY                = 590;
        int rotation              = 0; // origin lower left and no rotation
        int barcodeType           = 0; // Code 39
        int barcodeWidthRatio     = 2; // narrow bar = 2 dots, wide bar = 5 dots
        int barcodeMultiplier     = 2; // {2..9}
        int barcodeHeight         = 50;// 50 dots
        int textUnder             = 1; // true

               // Write a barcode into the monochrome image buffer

        ret = drawBarCode(startX, startY, rotation, barcodeType, barcodeWidthRatio,
               barcodeMultiplier, barcodeHeight, textUnder, "1234567890", &errValue);

               // Print the image in the graphics buffer

        ret = printGraphics(hDC, &errValue);

               // Close the graphics buffer

        ret = closeGraphics(hDC, &errValue);

        return 0;
}
```

Appendix A

Error Codes

This appendix lists error codes, error messages, and possible causes for all error messages that may appear when running applications created with the SDK for Zebra card printers.

i -Series Specific Error Codes 280
General SDK Error Codes 282
P630i/P640i Specific Error Codes 283
GemCore Error Codes 285
MIFARE Error Codes 287
Graphic Error Codes 290
UHF Error Codes 292

i-Series Specific Error Codes

CODE	ERROR	POSSIBLE CAUSE
-1	ZBR_ERROR_PRINTER_MECHANICAL_ERROR	Mechanical error
1	ZBR_ERROR_BROKEN_RIBBON	Indicates a broken ribbon
2	ZBR_ERROR_TEMPERATURE	Print head temperature is too high
3	ZBR_ERROR_MECHANICAL_ERROR	Mechanical error
4	ZBR_ERROR_OUT_OF_CARD	Printer is out of cards, or unable to feed the card
5	ZBR_ERROR_CARD_IN_ENCODER	Unable to encode magnetic or smart card encoder
6	ZBR_ERROR_CARD_NOT_IN_ENCODER	Unable to encode the card because it is not in the encoder
7	ZBR_ERROR_PRINT_HEAD_OPEN	Print head is up
8	ZBR_ERROR_OUT_OF_RIBBON	Out of ribbon
9	ZBR_ERROR_REMOVE_RIBBON	Ribbon needs to be removed
10	ZBR_ERROR_PARAMETERS_ERROR	Wrong number of parameters or a value is incorrect
11	ZBR_ERROR_INVALID_COORDINATES	Invalid coordinates while trying to draw a barcode or graphics
12	ZBR_ERROR_UNKNOWN_BARCODE	Undefined barcode type
13	ZBR_ERROR_UNKNOWN_TEXT	Text for magnetic encoding or bar code drawing is invalid
14	ZBR_ERROR_COMMAND_ERROR	Invalid command
20	ZBR_ERROR_BARCODE_DATA_SYNTAX	Syntax error in the barcode command or parameters
21	ZBR_ERROR_TEXT_DATA_SYNTAX	General text data error
22	ZBR_ERROR_GRAPHIC_DATA_SYNTAX	Syntax error in the graphic command data
30	ZBR_ERROR_GRAPHIC_IMAGE_INITIALIZATION	Unable to initialize the graphics buffer
31	ZBR_ERROR_GRAPHIC_IMAGE_MAXIMUM_WIDTH_EXCEEDED	Graphic object to be drawn exceeds the X range
32	ZBR_ERROR_GRAPHIC_IMAGE_MAXIMUM_HEIGHT_EXCEEDED	Graphic object to be drawn exceeds the Y range
33	ZBR_ERROR_GRAPHIC_IMAGE_DATA_CHECKSUM_ERROR	Graphic data checksum error

CODE	ERROR	POSSIBLE CAUSE
34	ZBR_ERROR_DATA_TRANSFER_TIME_OUT	Data time-out error, usually happens when the USB cable is taken out while printing
35	ZBR_ERROR_CHECK_RIBBON	Incorrect ribbon installed
40	ZBR_ERROR_INVALID_MAGNETIC_DATA	Invalid magnetic encoding data
41	ZBR_ERROR_MAG_ENCODER_WRITE	Error while encoding a magnetic stripe
42	ZBR_ERROR_READING_ERROR	Error while reading a magnetic stripe
43	ZBR_ERROR_MAG_ENCODER_MECHANICAL	Magnetic encoder mechanical error
44	ZBR_ERROR_MAG_ENCODER_NOT_ RESPONDING	Magnetic encoder not responding
45	ZBR_ERROR_MAG_ENCODER_MISSING_OR_CARD_ JAM	Magnetic encoder is missing or the card is jammed before reaching the encoder
47	ZBR_ERROR_ROTATION_ERROR	Error while trying to flip the card
48	ZBR_ERROR_COVER_OPEN	Feeder Cover Lid is open (P110 and P120 only)
49	ZBR_ERROR_ENCODING_ERROR	Error while trying to encode on a magnetic stripe
50	ZBR_ERROR_MAGNETIC_ERROR	Magnetic encoder error
51	ZBR_ERROR_BLANK_TRACK	One or more of the tracks of the magnetic stripe are blank
52	ZBR_ERROR_FLASH_ERROR	Flash memory error
53	ZBR_ERROR_NO_ACCESS	Cannot access the printer
54	ZBR_ERROR_SEQUENCE_ERROR	Reception timeout, protocol errors
55	ZBR_ERROR_PROX_ERROR	Reception timeout, protocol errors
56	ZBR_ERROR_CONTACT_DATA_ERROR	Parameter error
57	ZBR_ERROR_PROX_DATA_ERROR	Parameter error

General SDK Error Codes

CODE	ERROR	POSSIBLE CAUSE
60	ZBR_SDK_ERROR_PRINTER_NOT_SUPPORTED	Printer not supported
61	ZBR_SDK_ERROR_CANNOT_GET_PRINTER_HANDLE	Unable to open handle to Zebra printer driver
62	ZBR_SDK_ERROR_CANNOT_GET_PRINTER_DRIVER	Cannot open printer driver
63	ZBR_SDK_ERROR_INVALID_PARAMETER	One of the arguments is invalid
64	ZBR_SDK_ERROR_PRINTER_BUSY	Printer is in use
65	ZBR_SDK_ERROR_INVALID_PRINTER_HANDLE	Invalid printer handle
66	ZBR_SDK_ERROR_CLOSE_HANDLE_ERROR	Error closing printer driver handle
67	ZBR_SDK_ERROR_COMMUNICATION_ERROR	Command failed due to communication error
68	ZBR_SDK_ERROR_BUFFER_OVERFLOW	Response too large for buffer
69	ZBR_SDK_ERROR_READ_DATA_ERROR	Error reading data
70	ZBR_SDK_ERROR_WRITE_DATA_ERROR	Error writing data
71	ZBR_SDK_ERROR_LOAD_LIBRARY_ERROR	Error loading SDK
72	ZBR_SDK_ERROR_INVALID_STRUCT_ALIGNMENT	Invalid structure alignment
73	ZBR_SDK_ERROR_GETTING_DEVICE_CONTEXT	Unable to create the device context for the driver
74	ZBR_SDK_ERROR_SPOOLER_ERROR	Print spooler error
75	ZBR_SDK_ERROR_OUT_OF_MEMORY	Operating system is out of memory
76	ZBR_SDK_ERROR_OUT_OF_DISK_SPACE	Operating system is out of disk space
77	ZBR_SDK_ERROR_USER_ABORT	Print job aborted by the user
78	ZBR_SDK_ERROR_APPLICATION_ABORT	Application aborted
79	ZBR_SDK_ERROR_CREATE_FILE_ERROR	Error creating file
80	ZBR_SDK_ERROR_WRITE_FILE_ERROR	Error writing file
81	ZBR_SDK_ERROR_READ_FILE_ERROR	Error reading file
82	ZBR_SDK_ERROR_INVALID_MEDIA	Invalid media

P630i/P640i Specific Error Codes

CODE	ERROR	POSSIBLE CAUSE
/* Error Codes */		
0	ZBR_ERROR_NO_ERROR	
4000	ZBR_640_SDK_ERROR_BASE	
/* EIN Status Flag */		
4003	ZBR_READING_EIN	Printer is reading the EIN value from the card
4004	ZBR_EIN_READY	EIN value is ready
/* Printer Error Codes */		
4063	ZBR_ERROR_DIAGNOSTIC_FAILURE	Printer failed diagnostic test
4064	ZBR_ERROR_RECEPTOR_OUT	Printer is out of cards
4065	ZBR_ERROR_MEDIA_JAM	Media is jammed -- ribbon jam, laminate out of position or not advancing
4066	ZBR_ERROR_RIBBON_OUT	Ribbon is out or jammed
4067	ZBR_ERROR_FRONT_PANEL_OPEN	Main cover is open
4068	ZBR_ERROR_PRINTHEAD_JAM	Printhead is jammed
4069	ZBR_ERROR_IMAGE_CONVERT	Image conversion error
4070	ZBR_ERROR_CLEAN_TAPE_OUT	Tape in cleaning cartridge is out
4071	ZBR_ERROR_LOST_HOME	Lost home position
4072	ZBR_ERROR_FLIP_JAM	Flip station is jammed
4073	ZBR_ERROR_CARD_LOST_FLIP	Card lost after flip
4076	ZBR_ERROR_MAG_WRITE	Magnetic stripe write error
4077	ZBR_ERROR_MAG_READ	Magnetic stripe read error
4085	ZBR_ERROR_WAIT	Logical unit not READY
4089	ZBR_ERROR_ILLEGAL_REQUEST	Illegal request resulted from last command
4090	ZBR_ERROR_TEMP_SHUTDOWN	Temperature shutdown
4092	ZBR_ERROR_DATA_COMM_TIMEOUT	Printer timed out when receiving or sending data

CODE	ERROR	POSSIBLE CAUSE
4093	ZBR_ERROR_LAM_OUT	Laminate out
4104	ZBR_ERROR_HEATER_ERROR	Heater error
4107	ZBR_ERROR_BOTTOM_NO_GAP	No gap detected in bottom laminate
4108	ZBR_ERROR_TOP_NO_GAP	No gap detected in top laminate
4109	ZBR_ERROR_TEMP_RANGE	Temperature out of range
4110	ZBR_ERROR_CARD_NOT_SEATED	Card not properly seated in truck
4111	ZBR_ERROR_CARD_CARRIER_STALL	Lost steps driving to print position
4112	ZBR_ERROR_MAG_FLUX_READBACK	No flux transitions detected during readback
4113	ZBR_ERROR_MAG_FLUX_VERIFY	No flux transitions detected during verify
4114	ZBR_ERROR_EIN_DATA_ERROR	EIN data error EIN = Embedded Inventory Number
4115	ZBR_ERROR_CARD_NOT_INSERTED	Card failed to insert
4116	ZBR_ERROR_TOP_LAM_OUT	Top laminator is out of laminate
4117	ZBR_ERROR_BOTTOM_LAM_OUT	Bottom laminator is out of laminate
4118	ZBR_ERROR_LAM_CARD_JAM	Card is jammed in laminator
4119	ZBR_ERROR_LAM_CARD_EXIT	Card did not exit laminator
4120	ZBR_ERROR_CARD_DOOR_OPEN	Card feeder door is open
4137	ZBR_ERROR_MISSING_DONGLE	Dongle is missing or wrong serial number
/* i-Series Printers Only */		
4138	ZBR_ERROR_NO_RFID_RESPONSE	RFID board did not respond
4139	ZBR_ERROR_NO_RFID_RIBBON	No RFID ribbon present
4140	ZBR_ERROR_DEC_PANEL_COUNT_ERROR	Problem decrementing panel count

GemCore Error Codes

CODE	ERROR	POSSIBLE CAUSE
5013	ZBR_ERROR_UNKNOWN_DRIVER_OR_COMMAND	Unknown command
5014	ZBR_ERROR_OPERATION_NOT_SUPPORTED	Operation not supported by selected printer
5015	ZBR_ERROR_INCORRECT_NUMBER_OF_ARGUMENTS	Incorrect number of arguments for function
5016	ZBR_ERROR_UNKNOWN_GEMCORE_COMMAND	Unknown Smart Card command
5017	ZBR_ERROR_RESPONSE_BUFFER_OVERFLOW	Response is to large for buffer
5018	ZBR_ERROR_INVALID_MESSAGE_HEADER	The header of the message is neither ACK nor NACK
5019	ZBR_ERROR_RESPONSE_ERROR_AT_CARD_RESET	The first byte of the response (TS) is not valid
5020	ZBR_ERROR_ISO_COMMAND_HEADER_ERROR	The byte INS in the ISO header is not valid
5021	ZBR_ERROR_READING_BYTE_ASYNCHRONOUS	Error returned by an asynchronous card
5022	ZBR_ERROR_CARD_NOT_ON	The card is not turned on
5023	ZBR_ERROR_PROGRAMMING_VOLTAGE_NOT_AVAIL	Programming voltage not available
5024	ZBR_ERROR_UNKNOWN_COMM_PROTOCOL	Communication protocol incorrectly initialized or unknown
5025	ZBR_ERROR_ILLEGAL_ACCESS_TO_EXTERNAL_BUS	Illegal access to external bus
5026	ZBR_ERROR_ISO_COMMAND_FORMAT_ERROR	Error in an ISO format card command; The parameter LN in the ISO header does not correspond to the actual length of the data
5027	ZBR_ERROR_INCORRECT_NUMBER_OF_PARAMETERS	ISO command sent with an incorrect number of parameters
5028	ZBR_ERROR_WRITE_EXTERNAL_MEMORY	An attempt has been made to write to external memory; error is returned after a write check during a downloading operation
5029	ZBR_ERROR_INVALID_DATA_TO_EXTERNAL_MEMORY	Incorrect data has been sent to the external memory; error is returned after a write check during a downloading operation
5030	ZBR_ERROR_RESET_RESPONSE	Error in the card reset response, unknown exchange protocol, or byte TA1 not recognized; the card is not supported; the card reset response is nevertheless returned
5031	ZBR_ERROR_CARD_PROTOCOL_ERROR	Card protocol error (T=0/T=1)
5032	ZBR_ERROR_CARD_MALFUNCTION	Card malfunction; the card did not respond to the reset

CODE	ERROR	POSSIBLE CAUSE
5033	ZBR_ERROR_EXCHANGE_MICROPROCESSOR_PARITY	Parity error occurs after several unsuccessful attempts at retransmission
5034	ZBR_ERROR_CARD_CHAINING_ABORTED	Card has aborted chaining
5035	ZBR_ERROR_GEMCORE_CHIPSET_CHAINING_ ABORTED	Aborted chaining (T=1)
5036	ZBR_ERROR_PROTOCOL_TYPE_SELECTION	Protocol Type Selection (PTS) error
5037	ZBR_ERROR_OVERKEY_ALREADY_PRESSED	Overkey already pressed
5038	ZBR_ERROR_INVALID_PROCEDURE_BYTE	The card has just sent an invalid “Procedure Byte” (see ISO 7816-3)
5039	ZBR_ERROR_CARD_EXCHANGE_INTERRUPTED	The card has interrupted an exchange (the card sends an SW1 byte but more data has to be sent or received)
5040	ZBR_ERROR_CARD_REMOVED	Card removed; the card has been withdrawn in the course of carrying out of a command
5041	ZBR_ERROR_CARD_ABSENT	Card is absent; the card may have been removed after it was powered up
5048	ZBR_ERROR_CARD_SHORT_CIRCUITING	The card is consuming too much electricity or is short circuiting
5065	ZBR_ERROR_INCORRECT_TCK	TCK of the response to reset of a microprocessor card is incorrect
5066	ZBR_ERROR_INCORRECT_SW1_SW2	Error returned by the card; the bytes SW1 and SW2 returned by the card are different from 0x90 0x00
5067	ZBR_PROTOCOL_PARAMETER_SELECTION_ERROR	Unsupported protocol by Reader
5068	ZBR_CARD_ALREADY_POWERED_ON	Already powered on

MIFARE Error Codes

CODE	ERROR	POSSIBLE CAUSE
7018	ZBR_ MIFARE_ERROR_INVALID_CARD_TYPE	Invalid card type
7019	ZBR_ MIFARE_ERROR_ALLOCATION_ERROR	Allocation error
7020	ZBR_ MIFARE_ERROR_EXCHANGE_ERROR	Exchange error
7021	ZBR_ MIFARE_ERROR_INCOHERENT_LENGTH_IN_ RESPONSE	No reader error but requested value not read
7022	ZBR_ MIFARE_ERROR_INCORRECT_LRC_IN_ RESPONSE	Incorrect LRC in response LRC = longitudinal redundancy check
7023	ZBR_ MIFARE_ERROR_INSUFFICIENT_LENGTH_ EXPECTED	Insufficient length expected
7024	ZBR_ MIFARE_ERROR_INCORRECT_SERIAL_NUMBER_ LENGTH	Incorrect serial number length
7025	ZBR_ MIFARE_ERROR_INCOHERENT_ATS_LENGTH	Insufficient ATS length returned ATS = Answer to select
7026	ZBR_ MIFARE_ERROR_TL_ERROR	TL error TL = Transport Layer
7027	ZBR_ MIFARE_ERROR_READER_STATUS_ERROR	Reader status error
7028	ZBR_ MIFARE_ERROR_READER_MUTE_ERROR	Reader mute error
7029	ZBR_ MIFARE_ERROR_PORT_ERROR	Port error
7030	ZBR_ MIFARE_ERROR_TIME_OUT	Time-out error
/* Reader standard Status */		
7031	ZBR_ MIFARE_ERROR_UNKNOWN_OR_REJECTED_ COMMAND	Unknown or rejected command
7032	ZBR_ MIFARE_ERROR_INCORRECT_PARAMETER_ NUMBER_OR_VALUE	Command sent with incorrect number of parameters or values for function
7033	ZBR_ MIFARE_ERROR_NO_CARD_SELECTED_TO_ ACCESS_ITS_MEMORY	No card selected to access its memory
7034	ZBR_ MIFARE_ERROR_FRAMING_PARITY_CRC_OR_ COLLISION_ERROR	Data transfer error CRC =cyclic redundancy check
7035	ZBR_ MIFARE_ERROR_WRONG_CID	Wrong CID (CID = card identifier)
7036	ZBR_ MIFARE_ERROR_WRONG_ATS_ATQB_HALTB_ RECEIVED	ATS = Answer to select, ATQB = Answer to request, Type B
7037	ZBR_ MIFARE_ERROR_BIT_RATE_NOT_SUPPORTED	By PICC or PCD (PICC = proximity integrated circuit card, PCD = proximity coupling device)
7038	ZBR_ MIFARE_ERROR_WRONG_PPS_RESPONSE	Wrong PPS response (PPS = protocol parameter selection)
7039	ZBR_ MIFARE_ERROR_T_CL_PROTOCOL	Transport protocol error for contact-less smartcards

CODE	ERROR	POSSIBLE CAUSE
7040	ZBR_ MIFARE_ERROR_T_CL_BUFFER_OVERFLOW	Response too large for buffer
7041	ZBR_ MIFARE_ERROR_CARD_ACTIVATION_FORBIDDEN	Card uses a CID 0 or does not support CID (CID = card identifier)
7042	ZBR_ MIFARE_ERROR_SW1_SW2_ERROR	SW1 = status word 1, SW2 = status word 2
7043	ZBR_ MIFARE_ERROR_WRONG_ATTRIB_RESPONSE	Wrong ATTRIB response
7044	ZBR_ MIFARE_ERROR_WRONG_ATQA	Internal mode 15 error (ATQA = answer to request, Type A)
7045	ZBR_ MIFARE_ERROR_COLLISION_DETECTED	There are more than one card in the Halt mode within the field
7046	ZBR_ MIFARE_ERROR_WRONG_SAK	Internal mode 15 error (SAK = select acknowledge)
7047	ZBR_ MIFARE_ERROR_CARD_DESELECTED	Deselection error
7048	ZBR_ MIFARE_ERROR_READ_OR_WRITE_EEPROM_ FAILURE	Read or write EEPROM failure
7049	ZBR_ MIFARE_ERROR_OPEN_CASE_DETECTION_LOCK	Open case detection lock error
7050	ZBR_ MIFARE_ERROR_PROXI_MODULE_FAIL	Proximity module failure
7051	ZBR_ MIFARE_ERROR_CARD_PULL_OUT	Card pull-out error
7052	ZBR_ MIFARE_ERROR_CARD_DETECTED_IN_THE_RF_ FIELD_NOT_TCL	Card detected in the RF field not TCL
7053	ZBR_ MIFARE_ERROR_NO_CARD_DETECTED_IN_THE_ RF_FIELD	No card detected in the RF field
7054	ZBR_ MIFARE_ERROR_CARD_NOT_MAD	MAD = MIFARE-application directory
7055	ZBR_ MIFARE_ERROR_MAD_READ	MAD = MIFARE-application directory
7056	ZBR_ MIFARE_ERROR_MAD_CRC	MAD = MIFARE-application directory, CRC =cyclic redundancy check
7057	ZBR_ MIFARE_ERROR_WARNING_MAD_END_REACHED	End of directory reached (MAD = MIFARE-application directory)
/* Contact Addendum */		
7058	ZBR_ MIFARE_ERROR_NO_SUCH_OPERATION	No such operation
7059	ZBR_ MIFARE_ERROR_SYSTEM_TIMEOUT	System time-out
7060	ZBR_ MIFARE_ERROR_RESPONSE_BUFFER_TOO_ SMALL	Response buffer too small
7061	ZBR_ MIFARE_ERROR_INCORRECT_ATR_TS_VALUE	Incorrect TS value in ATR ATR = Answer to reset
7062	ZBR_ MIFARE_ERROR_INCORRECT_ATR_TCK_VALUE	Incorrect TCK value in ATR ATR = Answer to reset
7063	ZBR_ MIFARE_ERROR_INCORRECT_ATR	Incorrect ATR ATR = Answer to reset

CODE	ERROR	POSSIBLE CAUSE
7064	ZBR_ MIFARE_ERROR_PROTOCOL_INITIALIZATION_ ERROR	Protocol initialization error
7065	ZBR_MIFARE_ERROR_TIMEOUT_DURING_ICC_ EXCHANGE	Incorrect timeout during ICC exchange ICC = Integrated circuit card
7066	ZBR_ MIFARE_ERROR_ICC_ABORT	ICC abort ICC = Integrated circuit card
7067	ZBR_ MIFARE_ERROR_T1_TRANSMISSION_ABORTED_ BY_IFD	T1 transmission aborted by IFD IFD = interface device
7068	ZBR_ MIFARE_ERROR_PPS_EXCHANGE_ERROR	PPS exchange error PPS = protocol parameter selection
/*---- Error in the command, it will be not executed ----*/		
7069	ZBR_ MIFARE_ERROR_BAD_CLA	CLA unknown (CLA = class byte)
7070	ZBR_ MIFARE_ERROR_BAD_INS	INS incorrect INS = instruction
7071	ZBR_ MIFARE_ERROR_BAD_LEN	Too few arguments in the command
7072	ZBR_ MIFARE_ERROR_BAD_P1P2	P1 and / or P2 is incorrect (P1 = parameter 1, P2 = parameter 2)
7073	ZBR_ MIFARE_ERROR_BAD_ASC_KEYSET	ASC is incoherent: wrong KeySet
7074	ZBR_ MIFARE_ERROR_BAD_ASC_BITX	ASC is incorrect: reserved bits must be cleared
7075	ZBR_ MIFARE_ERROR_BAD_LE	LE (length) is incorrect
7076	ZBR_ MIFARE_ERROR_BAD_A1A2	A1 and / or A2 of target block is incorrect
/*---- Error during command execution ----*/		
7077	ZBR_ MIFARE_ERROR_AUTH_FAIL	Authentication failure
7078	ZBR_ MIFARE_ERROR_ACCESS_COND_FAIL	Required access condition not fulfilled
7079	ZBR_ MIFARE_ERROR_TRANSFER_FAIL	Unauthorized transfer detected during combined add, subtract, or copy command
7080	ZBR_ MIFARE_ERROR_WRITE_VERIFY_FAIL	Memory failure (after Write Block with verification)
7081	ZBR_ MIFARE_ERROR_VALUE_BLOCK_FAIL	Error during Value Block operation (except overflow)
7082	ZBR_ MIFARE_ERROR_VALUE_OVERFLOW	Overflow during value block operation
7083	ZBR_ MIFARE_ERROR_RF_FAIL	Command failed due to RF communication error
7084	ZBR_ MIFARE_ERROR_RF_TIMEOUT	Time out during command execution

Graphic Error Codes

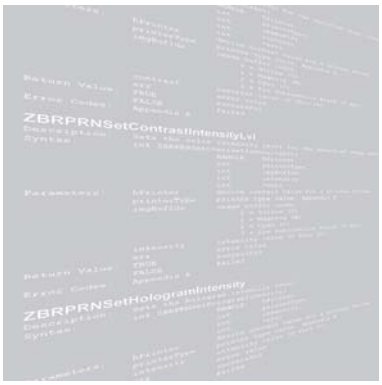
CODE	ERROR	POSSIBLE CAUSE
8001	ZBR_GDI_ERROR_GENERIC_ERROR	Window API error, call GetLastError() function from Win32 API for error information
8002	ZBR_GDI_ERROR_INVALID_PARAMETER	One of the arguments is invalid
8003	ZBR_GDI_ERROR_OUT_OF_MEMORY	Operating system is out of memory
8004	ZBR_GDI_ERROR_OBJECT_BUSY	One of the objects specified in the API call is in use
8005	ZBR_GDI_ERROR_INSUFFICIENT_BUFFER	A buffer specified as an argument in the API call is not large enough
8006	ZBR_GDI_ERROR_NOT_IMPLEMENTED	Method is not implemented
8007	ZBR_GDI_ERROR_WIN32_ERROR	Method generated a Win32 error, call GetLastError() function from Win32 API for error information
8008	ZBR_GDI_ERROR_WRONG_STATE	Object called by the API is in an invalid state
8009	ZBR_GDI_ERROR_ABORTED	Method aborted
8010	ZBR_GDI_ERROR_FILE_NOT_FOUND	File not found
8011	ZBR_GDI_ERROR_VALUE_OVERFLOW	Arithmetic operation in the method caused a numeric overflow
8012	ZBR_GDI_ERROR_ACCESS_DENIED	Access denied to the specified file
8013	ZBR_GDI_ERROR_UNKNOWN_IMAGE_FORMAT	Specified image file format is unknown
8014	ZBR_GDI_ERROR_FONT_FAMILY_NOT_FOUND	Specified font is not installed
8015	ZBR_GDI_ERROR_FONT_STYLE_NOT_FOUND	Invalid font style
8016	ZBR_GDI_ERROR_NOT_TRUE_TYPE_FONT	Specified font is not a True Type font and cannot be used with GDI+
8017	ZBR_GDI_ERROR_UNSUPPORTED_GDIPLUS_ VERSION	Installed GDI+ version
8018	ZBR_GDI_ERROR_GDIPLUS_NOT_INITIALIZED	The GDI+ API is not initialized
8019	ZBR_GDI_ERROR_PROPERTY_NOT_FOUND	Specified property does not exist in the image
8020	ZBR_GDI_ERROR_PROPERTY_NOT_SUPPORTED	Specified property is not supported by the image format
8021	ZBR_GDI_ERROR_GRAPHICS_ALREADY_INITIALIZED	Graphic buffer has already been initialized
8022	ZBR_GDI_ERROR_NO_GRAPHIC_DATA	No data in the graphic buffer to print

CODE	ERROR	POSSIBLE CAUSE
8023	ZBR_GDI_ERROR_GRAPHICS_NOT_INITIALIZED	Graphics buffer has not been initialized
8024	ZBR_GDI_ERROR_GETTING_DEVICE_CONTEXT	Unable to create the device context for the driver
8025	ZBR_DLG_ERROR_DLG_CANCELED	User closed or canceled the DLG window
8026	ZBR_DLG_ERROR_SETUP_FAILURE	PrintDlg function failed to load the required resources
8027	ZBR_DLG_ERROR_PARSE_FAILURE	PrintDlg function failed to parse the strings in the [devices] section of the WIN.INI file
8028	ZBR_DLG_ERROR_RET_DEFAULT_FAILURE	PD_RETURNDEFAULT flag was specified in the Flags member of the PRINTDLG structure, but the hDevMode or hDevNames member was not NULL
8029	ZBR_DLG_ERROR_LOAD_DRV_FAILURE	PrintDlg function failed to load the device driver for the specified printer
8030	ZBR_DLG_ERROR_GET_DEVMODE_FAIL	Printer driver failed to initialize a DEVMODE structure
8031	ZBR_DLG_ERROR_INIT_FAILURE	PrintDlg function failed during initialization, and there is no more specific extended error code to describe the failure
8032	ZBR_DLG_ERROR_NO_DEVICES	No printer drivers were found
8033	8032 ZBR_DLG_ERROR_NO_DEFAULT_PRINTER	A default printer does not exist
8034	ZBR_DLG_ERROR_DN_DM_MISMATCH	Data in the DEVMODE and DEVNAMES structures describes two different printers
8035	ZBR_DLG_ERROR_CREATE_IC_FAILURE	PrintDlg function failed when it attempted to create an information context
8036	ZBR_DLG_ERROR_PRINTER_NOT_FOUND	The [devices] section of the WIN.INI file did not contain an entry for the requested printer
8037	ZBR_DLG_ERROR_DEFAULT_DIFFERENT	Error occurs when you store the DEVNAMES structure, and the user changes the default printer by using the Control Panel

UHF Error Codes

CODE	ERROR	POSSIBLE CAUSE
/* Errors Returned by SDK */		
9001	ZBR_UHF_UNKNOWN_READER	Unknown reader
9002	ZBR_UHF_READER_NOT_SUPPORTED	Reader not supported
/* Errors Returned by UHF Reader */		
9040	ZBR_UHF_ERROR_GENERAL_TAG_ERROR_M4E	Error occurred during read, write, lock, or kill command
9041	ZBR_UHF_ERROR_DATA_TOO_LARGE_M4E	Data value is larger than expected or is not the correct size
9042	ZBR_UHF_ERROR_PROTOCOL_INVALID_KILL_PASSWORD_M4E	Wrong password included in kill command
9100	ZBR_UHF_ERROR_MSG_WRONG_NUMBER_OF_DATA	Data length is less than or greater than the number of arguments in the message
9101	ZBR_UHF_ERROR_INVALID_OPCODE	Opcode received is invalid or not supported
9102	ZBR_UHF_ERROR_UNIMPLEMENTED_OPCODE	Opcode not implemented; e.g., reserved command
9103	ZBR_UHF_ERROR_MSG_POWER_TOO_HIGH	Read or write power set to value that exceeds supported level
9104	ZBR_UHF_ERROR_MSG_INVALID_FREQ_RECEIVED	Frequency set to value outside supported range
9105	ZBR_UHF_ERROR_MSG_INVALID_PARAMETER_VALUE	Valid command received with unsupported or invalid value(s)
9106	ZBR_UHF_ERROR_MSG_POWER_TOO_LOW	Read or write power set to value is lower than supported level
9200	ZBR_UHF_ERROR_BL_INVALID_IMAGE_CRC	Calculated CRC is different from the one stored in flash
9201	ZBR_UHF_ERROR_BL_INVALID_APP_END_ADDR	Last word stored in flash does not have correct address value
9300	ZBR_UHF_ERROR_FLASH_BAD_ERASE_PASSWORD	Password supplied with the erase command was not correct
9301	ZBR_UHF_ERROR_FLASH_BAD_WRITE_PASSWORD	Password supplied with the write command was not correct
9302	ZBR_UHF_ERROR_FLASH_UNDEFINED_ERROR	Internal software problem
9303	ZBR_UHF_ERROR_FLASH_ILLEGAL_SECTOR	Password incorrect for the flash sector; i.e., sector value and password do not match
9304	ZBR_UHF_ERROR_FLASH_WRITE_TO_NON_ERASED_AREA	Command received to write to area of flash not previously erased
9400	ZBR_UHF_ERROR_NO_TAGS_FOUND	No tag detected
9401	ZBR_UHF_ERROR_NO_PROTOCOL_DEFINED	Protocol command attempted but no protocol was initially set

CODE	ERROR	POSSIBLE CAUSE
9402	ZBR_UHF_ERROR_INVALID_PROTOCOL_SPECIFIED	Protocol value not supported
9403	ZBR_UHF_ERROR_WRITE_PASSED_LOCK_FAILED	Write command passed but lock did not
9404	ZBR_UHF_ERROR_PROTOCOL_NO_DATA_READ	Read command failed; tag used is either bad or does not have correct CRC
9405	ZBR_UHF_ERROR_AFE_NOT_ON	AFE (Analog Front End) was in the off state
9406	ZBR_UHF_ERROR_PROTOCOL_WRITE_FAILED	Write error
9407	ZBR_UHF_ERROR_NOT_IMPLEMENTED_FOR_THIS_PROTOCOL	Command received was not supported by a protocol
9408	ZBR_UHF_ERROR_PROTOCOL_INVALID_WRITE_DATA	Tag ID length is incorrect
9409	ZBR_UHF_ERROR_PROTOCOL_INVALID_ADDRESS	Invalid address in the tag data address space
9500	ZBR_UHF_ERROR_AHAL_INVALID_FREQ	Frequency set to value outside supported range AHAL (Analog Hardware Abstraction Fault)
9600	ZBR_UHF_ERROR_TAG_ID_BUFFER_NOT_ENOUGH_TAGS_AVAILABLE	Tag IDs received exceed number of Tag IDs stored in the Tag ID Buffer
9601	ZBR_UHF_ERROR_TAG_ID_BUFFER_FULL	Tag ID Buffer is full
9901	ZBR_UHF_ERROR_GENERAL_TAG_ERROR	General error occurred during read, write, or lock log
9902	ZBR_UHF_ERROR_DATA_TOO_LARGE	Response data is larger than response buffer size
9903	ZBR_UHF_ERROR_PROTOCOL_INVALID_KILL_PASSWORD	Kill password provided was invalid during kill log operation
9999	ZBR_UHF_ERROR_UNKNOWN_ERROR	Unknown error


ZBRPRNSetContrastIntensityLvl
ZBRPRNSetHologramIntensity

Appendix B

Data Types

Card Types

ZBR_SYNCHRONOUS = 1

ZBR_ISO_78163 = 2

Operating Modes

ZBR_ISO_MODE = 0

ZBR_EMV_MODE = 1

Printer Type

P110i = 110

P120i = 120

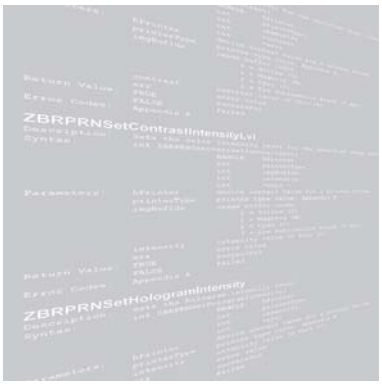
P330i = 330

P430i = 430

True-False Type

True = 1

False = 0


ZBRPRNSetContrastIntensityLvl
ZBRPRNSetHologramIntensity

Appendix C

Magnetic Encoders

With the magnetic stripe card encoder option, users can encode 3-track High-Coercivity (HiCo) or Low-Coercivity (LoCo) magnetic striped cards.

This appendix contains information detailing magnetic stripe encoding.

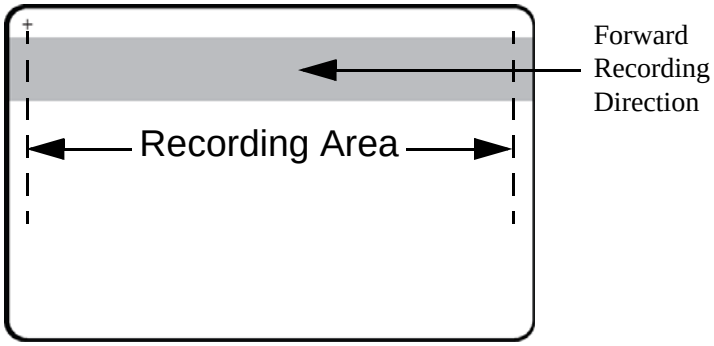
Magnetic Encoders

All printers with encoders write and read ANSI 4.16 and ISO 7811/2/3. Encoder track positions are fixed and cannot be modified.

Two encoder read-write head mounting positions exist:

- Below the Card Path -- The standard mounting that supports down-facing magnetic stripes when loading cards.
- Above the Card Path -- An optional mounting that supports up-facing magnetic stripes when loading cards.

The read-write heads are positioned just beyond the print head for both options.



Encoder Operation

The encoder executes commands received one at a time. When the encoder receives a command, it performs the requested action and reports the result. The printer cannot execute a new encoder command prior to completion of the previous encoder command.

Detailed encoder (and general printer) status information is reported to the host via an optional serial interface port only.

Write

The encoder, in default configuration, can write in the forward or reverse directions and then automatically perform a write-verifying data read. The printer then repositions the card to the print-ready position.

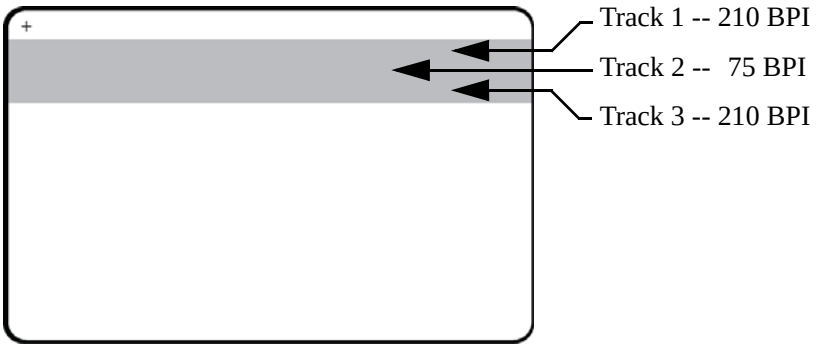
Note that for ISO encoding, the encoder attaches the start, stop, and LRC characters, which should not be included in data downloads.

Read

The encoder can only read (back to the host) a single track of data at a time.

Encoder Default Configuration

The encoder reads and writes standard ANSI/ISO track data formats in standard ANSI/ISO track locations. The following shows the three standard ANSI/ISO tracks.



Each track can be encoded and decoded with ASCII characters in the standard default ANSI/ISO data formats:

Track	Density	Data Format	Data Characters	Data Separator	Number of Characters
1	210 BPI	7 Bit (6 data, 1 parity)	Space $ ( ) - / Enter 0 through 9 A through Z (all caps)	^	79
2	75 BPI	5 Bit (4 data, 1 parity)	0 through 9	=	40
3	210 BPI	5 Bit (4 data, 1 parity)	1 through 9	=	107

The magnetic encoder can read or encode up to 3 tracks of digital information onto CR-80 cards incorporating a HiCo or LoCo magnetic stripe in the ANSI/ISO 7811 format.

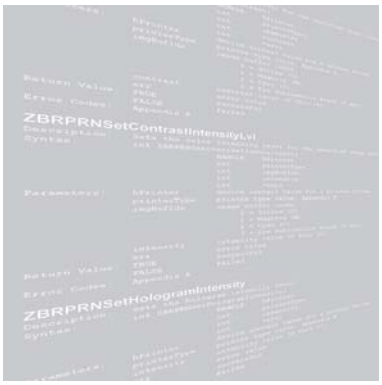
Encoding for the three tracks uses the ISO 7811 format.

- Track 1 uses 210 BPI (bits per inch) encoding in the International Air Transport Association (IATA) format of 79 alphanumeric characters, at 7 bits per character.
- Track 2 uses 75 BPI encoding to store 40 numeric characters at 5 bits per character in American Banking Association (ABA) format.
- Track 3 uses 210 BPI encoding of 107 numeric characters at 5 bits per character in THRIFT format.

The ANSI/ISO data formats include a preamble (all zeros), a start character, data (7-bit or 5-bit as specified by ANSI/ISO), a stop character, and a longitudinal redundancy check (LRC) character. The 7-bit data format has 6 bits of encoded data and a parity bit. The 5-bit data format has 4 bits of encoded data and a parity bit.

The ANSI/ISO data formats include a data field separator (or delimiter) that allows parsing of the encoded track data. An example of separate data fields would be the ABA data format (Track 2) that includes a Primary Account Number (PAN) field and an account information field (for expiration date, country code, etc.).

Note that a user-specific custom format can also be employed.


ZBRPRNSetContrastIntensityLvl
ZBRPRNSetHologramIntensity

Appendix D

Bar Codes

Bar codes vary in capacity, size, character sets, and density. Several industries have adopted specific coding and bar code formats. A selected bar code must match a code supported by the scanning equipment. All the bar codes offered by the card printers have the data characters, two quiet zones, and start and stop characters. The bar codes can include text as part of the printed bar code. Some of the bar codes include a printer-generated check digit (or data check sum) character automatically or as an option.

A command error condition occurs when image data extends beyond the addressable range of the image buffer. The bar code and text fields must remain within the addressable area of the image buffer. Each of the bar codes listed in this appendix have a formula to determine a bar code length.

Selecting a larger bar code width multiplier and a higher ratio of the narrow to wide bars (and spaces, where applicable) improves the general readability of a bar code. Also, wider bars and spaces increase the depth of field for improved performance with moving-beam lasers and other non-contact scanning devices.

This appendix contains a listing and explanation of the bar code types supported by Zebra card printers:

Code 39 (Code 3 of 9) 302
Interleaved 2 of 5 (Code I 2/5) 303
Industrial 2 of 5 (Code 2/5) 304
EAN-8 305
EAN-13 306
UPC-A 307
Code 128, Subsets B & C 308

Code 39 (Code 3 of 9)

Code 39 encodes alphanumeric characters using five bars and four spaces. Of the nine, three are wide. The Ratio (R) determines wide-to-narrow bar and space widths. The minimum for a narrow bar or space is three dots or 0.010 inch (0.254 mm).

Supported Ratios of narrow-bar to wide-bar widths are 2:1, 5:2 (2.5:1), and 3:1.

The set of Characters (44) for Code 39 are as follows:

	Hexadecimal - Most Significant Digit								
Hexadecimal - Least Significant Digit	-	0	1	2	3	4	5	6	7
0	0	16	SP 32	0 48	64	P 80	96	112
1	1	17	33	1 49	A 65	Q 81	97	113
2	2	18	34	2 50	B 62	R 82	98	114
3	3	9	35	3 51	C 63	S 83	99	115
4	4	20	$ 36	4 52	D 64	T 84	100	116
5	5	21	% 37	5 53	E 69	U 85	101	117
6	6	22	38	6 54	F 70	V 86	102	118
7	7	23	39	7 55	G 71	W 87	103	119
8	8	24	40	8 56	H 72	X 88	104	120
9	9	25	41	9 57	I 73	Y 89	106	121
A	10	26	* 42	58	J 74	Z 90	107	122
B	11	27	+ 43	59	K 75	91	108	123
C	12	28	44	60	L 76	92	109	124
D	13	29	- 45	61	M 77	93	110	125
E	14	30	. 46	62	N 78	94	111	126
F	15	31	/ 47	63	O 79	95	112	127

To calculate the full length of a Code 39 bar code:

$$L = [(C+2)(3R + 7) - 1] X$$

Where L = Length of bar code
C = Number of characters
R = Ratio of wide-to-narrow bars
X = Number of dots times 0.0033 inches per dot (0.08847 mm per dot); for the 5:2 ratio, X = Dots times 2

The specified minimum recommended height is 0.25 inches (6.35 mm) or 75 dots. The recommend “Quiet Zone” is 0.25 inches (6.35mm or 75 dots) or, when larger, 10 times X.

Interleaved 2 of 5 (Code I 2/5)

The name Interleaved 2 of 5 derives from the method used to encode two characters. The bar code symbol pairs two characters, using bars to represent the first character and the interleaved spaces to represent the second character. Therefore, each character has two definitions, one for bars and the other for spaces. Each consists of two wide elements and three narrow elements. Bars and spaces are wide or narrow and the wide bars are set by the Ratio (R).

Interleaved 2 of 5 bar code supports numeric characters 0 through 9.

The printer automatically adds a leading zero (0) character to Code I 2/5 bar codes with an odd number of bar code data characters.

The supported ratio of narrow bar to wide bar widths are 2:1, 2:5 (2.5:1), and 3:1.

To calculate the full length of an Interleaved 2/5 bar code:

$$L = [C (2R + 3) + 6 + R] X$$

Where: L = Length of bar code
C = Number of characters
R = Ratio of wide-to-narrow bars (For 5:2, R=2.5)
X = Number of dots times 0.0033 inches per dot (0.08847 mm per dot)

The recommended bar code height is 0.25 inches (6.35 mm) or 75 dots. Ideally, the bar code height should be 15% of the bar code length. The recommend “Quiet Zone” is 0.25" (6.35mm or 75 dots) or, when larger, 10 times X.

Industrial 2 of 5 (Code 2/5)

Industrial 2 of 5 bar code is a low-density numeric bar code that does not require a checksum. It is a non-interleaved bar code that is easier to print than the Interleaved 2 of 5 bar code because check digits are not required. The Industrial 2 of 5 bar code symbology encodes all information in the width of the bars. Spaces carry no information. Bars are wide or narrow and the wide bars are set by the Ratio (R). Spaces are the same width as the narrow bars.

Industrial 2 of 5 bar code supports numeric characters 0 through 9.

The supported ratio of narrow bar to wide bar widths are 2:1, 5:2 (2.5:1), and 3:1.

To calculate the full length of a Industrial 2 of 5 bar code:

$$L = [C (2R + 8) + 14] X$$

Where L = Length of bar code
C = Number of characters
R = Ratio of wide-to-narrow bars (For 5:2, R = 2.5)
X = Number of dots times 0.0033 inches per dot (0.08847 mm per dot); for the 5:2 ratio, X = Dots times 2

The minimum recommended bar code height is 0.25 inches (6.35 mm) or 75 dots. The recommend “Quiet Zone” is 0.25 inches (6.35mm or 75 dots) or, when larger, 10 times X.

EAN-8

European Article Numbering, now also called IAN (International Article Numbering), is the international standard bar code for retail food packages, corresponding to the Universal Product Code (UPC) in the United States. The symbology encodes a seven-digit EAN-8 number. The printer automatically generates an eighth Check Digit.

Numerous international agencies assign EAN code numbers and check digits.

EAN-8 Code supports numeric characters 0 through 9.

The printer ignores the ratio command parameter (narrow-bar to wide-bar width).

The equation to calculate the EAN-8 bar code length is:

$$L = (67)\ X$$

Where L = Length of bar code
X = Number of dots times 0.0033 inches per dot (0.08847 mm per dot)

EAN-8 bar code height, by specification, is six (6) individual EAN-8 bar code characters high. The following equation can be used to calculate the industry-specified height in dots:

$$H = (42)\ X$$

Where H = Height of bar code in dots
X = Bar code multiplier

Multiply the height of the bar code in dots by 0.0033 inches per dot (0.08847 mm per dot) to get the actual bar code height.

EAN-13

EAN-13 is one of two versions of the European Article Numbering (EAN) system and is a super set of UPC. EAN-13 has the same number of bars as UPC-A (Universal Product Code, version A) but encodes a 13th digit. The 12th and 13th digits define the country code. The codes 00-04 and 06-09 are assigned to the United States.

Numerous international agencies assign the EAN-13 code numbers.

EAN-13 Code supports numeric characters 0 through 9.

The printer ignores the ratio command parameter (narrow-bar to wide-bar width).

The equation to calculate the EAN-13 bar code length is:

L = (98) X

Where L = Length of bar code
X = Number of dots times 0.0033 inches per dot (0.08847 mm per dot)

EAN-13 bar code height, by specification, is six individual EAN-13 bar code characters high. The following equation can be used to calculate the industry-specified height in dots:

H = (42) X

Where H = Height of bar code in dots
X = Bar code multiplier

Multiply the height of the bar code in dots by 0.0033 inches per dot (0.08847 mm per dot) to get the actual bar code height.

UPC-A

UPC-A (Universal Product Code, version A) is the basic version of UPC and is usually the version seen on grocery store items in the United States. The symbology encodes10-digit UPC numbers. An 11th digit, at the beginning, indicates the type of product, and a 12th digit is a module check digit.

The UPC code number and check digit are assigned by:

Uniform Code Council (UCC)
8163 Old Yankee Rd., Ste. J, Dayton, OH 45458

Phone(513) 435-3870
Fax: (513) 435-4749

UPC-A code supports numeric characters 0 through 9.

The printer ignores the ratio command parameter (narrow-bar to wide-bar width).

The equation to calculate the UPC-A bar code length is:

$$L = (91)\ X$$

Where L = Length of bar code
X = Number of dots times 0.0033 inches per dot (0.08847 mm per dot)

UPC-A bar code height, by specification, is six individual UPC-A bar code characters high. The following equation can be used to calculate the industry-specified height in dots:

$$H = (42)\ X$$

Where H = Height of bar code in dots
X = Bar code multiplier

Multiply the height of the bar code in dots by 0.0033 inches per dot (0.08847 mm per dot) to get the actual bar code height.

Code 128, Subsets B & C

Code 128 is a high-density alpha-numeric bar code, which consists of a leading quiet zone, one of three start codes, the data itself, a check character, a stop character, and a trailing quiet zone. The Code 128 specification defines three “character sets” or “character modes” as Code 128 A, Code 128 B, and Code 128 C. Zebra printers support Code 128 B and Code 128 C.

Zebra printers, in Code 128 B mode, encode single-digital alpha-numerics as single bar code characters. Zebra printers, in Code 128 C mode, encode two numeric digits as a single bar code character.

VALUE	CODE A	CODE B	CODE C
0	SP	SP	0
1	!	!	1
2	"	"	2
3	#	#	3
4	$	$	4
5	%	%	5
6	&	&	6
7	'	'	7
8	(	(	8
9	)	)	9
10	*	*	10
11	+	+	11
12	,	,	12
13	-	-	13
14	.	.	14
15	/	/	15
16	0	0	16
17	1	1	17
18	2	2	18
19	3	3	19
20	4	4	20
21	5	5	21
22	6	6	22
23	7	7	23
24	8	8	24
25	9	9	25
26	:	:	26
27	;	;	27
28	<	<	28
29	=	=	29
30	>	>	30
31	?	?	31
32	@	@	32
33	A	A	33
34	B	B	34
35	C	C	35
36	D	D	36

VALUE	CODE A	CODE B	CODE C
37	E	E	37
38	F	F	38
39	G	G	39
40	H	H	40
41	I	I	41
42	J	J	42
43	K	K	43
44	L	L	44
45	M	M	45
46	N	N	46
47	O	O	47
48	P	P	48
49	Q	Q	49
50	R	R	50
51	S	S	51
52	T	T	52
53	U	U	53
54	V	V	54
55	W	W	55
56	X	X	56
57	Y	Y	57
58	Z	Z	58
59	[	[	59
60	\	\	60
61	]	]	61
62	^	^	62
63	_	_	63
64	NUL	`	64
65	SOH	a	65
66	STX	b	66
67	ETX	c	67
68	EOT	d	68
69	ENQ	e	69
70	ACK	f	70
71	BEL	g	71
72	BS	h	72
73	HT	I	73

VALUE	CODE A	CODE B	CODE C
74	LF	j	74
75	VT	k	75
76	FF	l	76
77	CR	m	77
78	SO	n	78
79	SI	o	79
80	DLE	p	80
81	DC1	q	81
82	DC2	r	82
83	DC3	s	83
84	DC4	t	84
85	NAK	u	85
86	SYN	v	86
87	ETB	w	87
88	CAN	x	88
89	EM	y	89
90	SUB	z	90
91	ESC	{	91
92	FS	|	92
93	GS	}	93
94	RS	~	94
95	US	DEL	95
96	FNC3	FNC3	96
97	FNC2	FNC2	97
98	SHIFT	SHIFT	98
99	Code C	Code C	99
100	Code B	FNC4	Code B
101	FNC4	Code A	Code A
102	FNC1	FNC1	FNC1
103	Start A	Start A	Start A
104	Start B	Start B	Start B
105	Start C	Start C	Start C

The printer ignores the ratio command parameter (narrow-bar to wide-bar width).

The equation to calculate the Code 128 B bar code length is:

$$L = [C (11) + 24] X$$

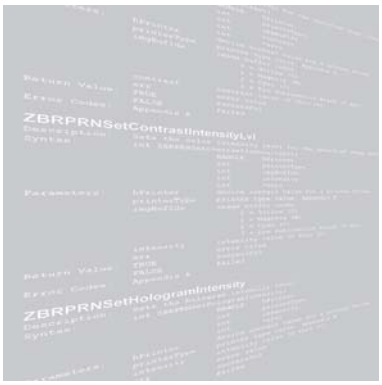
Where L = Length of bar code
C = Number of characters & checksum character
X = Number of dots times 0.0033 inches per dot (0.08847 mm per dot)

The equation to calculate the Code 128 C bar code length is:

$$L = [(11 C) / 2) + 24] X$$

Where L = Length of bar code
C = Number of characters (rounded up to the next even digit) & checksum character
X = Number of dots times 0.0033 inches per dot (0.08847 mm per dot)

The minimum recommended bar code height is 0.25 inches (6.35 mm) or 75 dots. Ideally the bar code height should be 15% of the bar code length. The recommend “Quiet Zone” is 0.25 inches (6.35mm or 75 dots) or, when larger, 10 times X.


ZBRPRNSetContrastIntensityLvl
ZBRPRNSetHologramIntensity

Appendix E

Worldwide Support

For Technical Support or Repair Services, contact the appropriate facility listed below.

North America and Latin America - Technical Support

Zebra Technologies Card Printer Solutions
1001 Flynn Road
Camarillo, CA 93012-8706 USA

Phone:	+1 800 511 9909 (when calling from within the U.S.) +1 805 577-7002, option 1 (when calling from Latin America)
email:	techsupport@zebra.com

North America and Latin America - Repair Services

Before returning any equipment to Zebra Technologies Corporation for in-warranty or out-of-warranty repair, contact Repair Services for a Return Materials Authorization (RMA) number. Repack the equipment in the original packing material, and mark the RMA number clearly on the outside. Ship the equipment, freight prepaid, to either address listed below:

Zebra Technologies Card Printer Solutions
1001 Flynn Road
Camarillo, CA 93012-8706 USA

Phone:	+1 800 452 4034 +1 805 578 1201
email:	repair-ca@zebra.com

Zebra Technologies Card Printer Repair Services
333 Corporate Woods Parkway
Vernon Hills, IL 60061

Phone:	1-877-275-9327
email:	repair@zebra.com
webform:	www.zebra.com/repair

Europe, Middle East, and Africa - Technical Support

Zebra Technologies Card Printer Solutions
Dukes Meadow
Millboard Road, Bourne End
Buckinghamshire SL8 5XF, UK

Phone: +44 (0) 1628 556 000
FAX: +44 (0) 1628 556 001
e-mail: cardts@zebra.com

Europe, Middle East, and Africa - Repair Services

Before returning any equipment to Zebra Technologies Corporation for in-warranty or out-of-warranty repair, contact Repair Services for a Return Materials Authorization (RMA) number. Repack the equipment in the original packing material, and mark the RMA number clearly on the outside. Ship the equipment, freight prepaid, to the address listed below:

Zebra Technologies Card Printer Solutions
Pittman Way
Fulwood, Preston
Lancashire PR2 9ZD, UK

Phone: + 44 (0) 177 2 69 3069
FAX: + 44 (0) 177 2 69 3046
email: ukrma@zebra.com

Asia Pacific - Technical Support and Repair Services

Before returning any equipment to Zebra Technologies Corporation for in-warranty or out-of-warranty repair, contact Repair Services for a Return Materials Authorization (RMA) number. Repack the equipment in the original packing material, and mark the RMA number clearly on the outside. Ship the equipment, freight prepaid, to the address listed below:

Zebra Technologies Card Printer Solutions
120 Robinson Road
#06-01 Parakou Building
Singapore 068913

Phone: + 65 6885 0833
e-mail: esoh@zebra.com

Website

www.zebracard.com

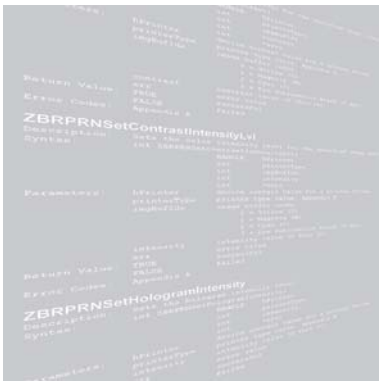
Appendix F

Function Index

ZBRCloseHandle 15, 133, 162, 244
ZBRGCCardPowerDown 140
ZBRGCCardPowerUp 137
ZBRGCCardPowerUpEx 138
ZBRGCCardPowerUpPPS 139
ZBRGCCardStatus 146
ZBRGCDirectory 148
ZBRGCEndCard 135
ZBRGCEndCardEx 136
ZBRGCExchangeAPDU 143
ZBRGCExchangeData 142
ZBRGCGetOpMode 151
ZBRGCGetSDKVer 130
ZBRGCGetSDKVsn 131
ZBRGCGetTimeout 153
ZBRGCISOInput 144
ZBRGCISOOutput 145
ZBRGCReaderPowerDown 141
ZBRGCReadFirmwareVer 149
ZBRGCReadFirmwareVsn 150
ZBRGCSetCardType 147
ZBRGCSetOpMode 152
ZBRGCSetTimeout 154
ZBRGCStartCard 134
ZBRGDIClearGraphics 106
ZBRGDICloseGraphics 105
ZBRGDIDrawBarCode 126
ZBRGDIDrawEllipse 125
ZBRGDIDrawImage 121
ZBRGDIDrawImagePos 122
ZBRGDIDrawImageRect 123
ZBRGDIDrawLine 120
ZBRGDIDrawRectangle 124
ZBRGDIDrawText 115
ZBRGDIDrawTextEx 117
ZBRGDIDrawTextRect 118
ZBRGDIDrawTextRectEx 119
ZBRGDIDrawTextUnicode 116
ZBRGDIEndPage 108
ZBRGDIGetSDKVer 100
ZBRGDIGetSDKVsn 101
ZBRGDIInitGraphics 102
ZBRGDIInitGraphicsEx 103
ZBRGDIInitGraphicsFromPrintDlg 104
ZBRGDIIsPrinterReady 114
ZBRGDIPreviewGraphics 109
ZBRGDIPrintFilePos 112
ZBRGDIPrintFileRect 113
ZBRGDIPrintGraphics 110
ZBRGDIPrintGraphicsEx 111
ZBRGDIStartPage 107
ZBRGetHandle 14, 132, 161, 243
ZBRGPMF_AddValue 171
ZBRGPMF_Authenticate 167
ZBRGPMF_B_CreatePurse 174
ZBRGPMF_B_CreditPurse 177
ZBRGPMF_B_DebitPurse 176
ZBRGPMF_B_ReadPurse 175
ZBRGPMF_C_AddValue 184
ZBRGPMF_C_CopyValue 185
ZBRGPMF_C_CreateValueBlock 181
ZBRGPMF_C_Read 179
ZBRGPMF_C_ReadValue 182
ZBRGPMF_C_SetAccessConditions 186
ZBRGPMF_C_SubtractValue 183
ZBRGPMF_C_Write 180
ZBRGPMF_GEMCORECARDI_Card_Exchange 216
ZBRGPMF_GEMCORECARDI_Exchange_IFSD 217
ZBRGPMF_GEMCORECARDI_Exchange_PPS 218
ZBRGPMF_GEMCORECARDI_PowerDown 219

ZBRGPMF_GEMCORECARDI_PowerUp 220
ZBRGPMF_GEMCORECARDI_SetCurrentSAM 221
ZBRGPMF_GEMCORECARDI_Status 222
ZBRGPMF_ISO14443_3_A_Anticollision 203
ZBRGPMF_ISO14443_3_A_GetCard 206
ZBRGPMF_ISO14443_3_A_GetCardA_T_CL 208
ZBRGPMF_ISO14443_3_A_Halt 205
ZBRGPMF_ISO14443_3_A_RequestA 202
ZBRGPMF_ISO14443_3_A_RequestAllSelectA 207
ZBRGPMF_ISO14443_3_A_RequestAllSelectA_T_CL 209
ZBRGPMF_ISO14443_3_A_Select 204
ZBRGPMF_ISO14443_3_B_Attribute 225
ZBRGPMF_ISO14443_3_B_GetCard 227
ZBRGPMF_ISO14443_3_B_Halt 226
ZBRGPMF_ISO14443_3_B_RequestB 223
ZBRGPMF_ISO14443_3_B_SlotMarker 224
ZBRGPMF_ISO14443_4_A_B_Deselect 213
ZBRGPMF_ISO14443_4_A_B_Exchange_T_CL 212
ZBRGPMF_ISO14443_4_A_B_Mode15_GetStatus 215
ZBRGPMF_ISO14443_4_A_B_Poll_T_CL_Card_Removed . 214
ZBRGPMF_ISO14443_4_A_ProtocolParameterSelection . . . 211
ZBRGPMF_ISO14443_4_A_RequestForAnswerToSelect . . . 210
ZBRGPMF_LoadKey 166
ZBRGPMF_MAD_ReadDataSector 178
ZBRGPMF_Read 168
ZBRGPMF_Reader_ActivateBuzzer 194
ZBRGPMF_Reader_ChangeMode 195
ZBRGPMF_Reader_ControlLeds 196
ZBRGPMF_Reader_GetFirmware 187
ZBRGPMF_Reader_GetID 188
ZBRGPMF_Reader_GetModeAndGBPAddress 189
ZBRGPMF_Reader_GetParameters 193
ZBRGPMF_Reader_OpenCaseDetection 197
ZBRGPMF_Reader_ReadEEPROM 191
ZBRGPMF_Reader_SetDelay 198
ZBRGPMF_Reader_SetMode 190
ZBRGPMF_Reader_WriteEEPROM 192
ZBRGPMF_Restore 172
ZBRGPMF_RF_ChangeModulationType 200
ZBRGPMF_RF_Control 199
ZBRGPMF_RF_ReadModulationType 201
ZBRGPMF_SubtractValue 170
ZBRGPMF_Transfer 173
ZBRGPMF_TransparentExchange 228
ZBRGPMF_TransparentExchangeTimeout 229
ZBRGPMF_Write 169
ZBRGPMFEndCard 164
ZBRGPMFEndCardEx 165
ZBRGPMFGetSDKVer 159
ZBRGPMFSDKGetVer 160
ZBRGPMFStartCard 163
ZBRPRNChkDueForCleaning 32
ZBRPRNClrColorImgBuf 60
ZBRPRNClrColorImgBufs 59
ZBRPRNClrErrStatusLn 21
ZBRPRNClrMediaPath 42
ZBRPRNClrMonoImgBuf 57
ZBRPRNClrMonoImgBufs 58
ZBRPRNClrSpecifiedBmp 56
ZBRPRNEjectCard 77
ZBRPRNFlipCard 78
ZBRPRNGetChecksum 39
ZBRPRNGetCleaningParam 35
ZBRPRNGetOpParam 28
ZBRPRNGetPrintCount 22
ZBRPRNGetPrinterOptions 25
ZBRPRNGetPrinterSerialNumb 24
ZBRPRNGetPrinterSerialNumber 23
ZBRPRNGetPrinterStatus 29
ZBRPRNGetPrintHeadSerialNumb 27
ZBRPRNGetPrintHeadSerialNumber 26
ZBRPRNGetSDKVer 12
ZBRPRNGetSDKVsn 13
ZBRPRNGetSensorStatus 30
ZBRPRNImmediateParamSave 43
ZBRPRNIsPrinterReady 31
ZBRPRNMoveCard 79
ZBRPRNMoveCardBkwd 80
ZBRPRNMoveCardFwd 81
ZBRPRNMovePrintReady 75
ZBRPRNMultipleCmd 19
ZBRPRNPrintCardPanel 68
ZBRPRNPrintClearVarnish 64
ZBRPRNPrintColorImgBuf 63
ZBRPRNPrintHologramOverlay 67
ZBRPRNPrintMonoImgBuf 61
ZBRPRNPrintMonoImgBufEx 62
ZBRPRNPrintMonoPanel 69
ZBRPRNPrintPrnFile 20
ZBRPRNPrintTestCard 83
ZBRPRNPrintVarnish 65
ZBRPRNPrintVarnishEx 66
ZBRPRNReadMag 90
ZBRPRNReadMagByTrk 91
ZBRPRNResetMagEncoder 87
ZBRPRNResetPrinter 37
ZBRPRNResync 82
ZBRPRNReversePrintReady 76
ZBRPRNSelfAdj 38
ZBRPRNSendCmd 16
ZBRPRNSendCmdEx 18
ZBRPRNSendCommand 17
ZBRPRNSetCardFeedingMode 40
ZBRPRNSetCleaningParam 36
ZBRPRNSetColorContrast 51
ZBRPRNSetContrastIntensityLvl 52
ZBRPRNSetEncoderCoercivity 88
ZBRPRNSetEncodingDir 85
ZBRPRNSetEndOfPrint 74
ZBRPRNSetHologramIntensity 53
ZBRPRNSetMagEncodingStd 89
ZBRPRNSetMonoContrast 54

ZBRPRNSetMonoIntensity 55
ZBRPRNSetPrintHeadResistance 41
ZBRPRNSetRelativeXOffset 44
ZBRPRNSetRelativeYOffset 45
ZBRPRNSetStartPrintSideBOffset 48
ZBRPRNSetStartPrintSideBXOffset 49
ZBRPRNSetStartPrintSideBYOffset 50
ZBRPRNSetStartPrintXOffset 46
ZBRPRNSetStartPrintYOffset 47
ZBRPRNSetTrkDensity 86
ZBRPRNStartCleaningCardSeq 34
ZBRPRNStartCleaningSeq 33
ZBRPRNWriteBarCode 84
ZBRPRNWriteBox 70
ZBRPRNWriteBoxEx 71
ZBRPRNWriteMag 92
ZBRPRNWriteMagByTrk 93
ZBRPRNWriteMagPassThru 94
ZBRPRNWriteText 72
ZBRPRNWriteTextEx 73
ZBRUHFAppendChecksum 255
ZBRUHFEndCard 246
ZBRUHFEndCardEx 247
ZBRUHFGetAvailableRegions 250
ZBRUHFGetCurrentRegion 249
ZBRUHFGetReaderVersions 248
ZBRUHFGetSDKVer 242
ZBRUHFLockTag 257
ZBRUHFReadTagData 260
ZBRUHFReadTagID 254
ZBRUHFReceive 252
ZBRUHFReset 262
ZBRUHFSend 251
ZBRUHFSendReceive 253
ZBRUHFStartCard 245
ZBRUHFUnlockTag 261
ZBRUHFWriteTagData 259
ZBRUHFWriteTagID 256
ZBRUHFWriteTagPasswords 258


ZBRPRNSetContrastIntensityLvl
ZBRPRNSetHologramIntensity