Return Value: contrast err TRUE FALSE
Error Codes: Appendix A
0 = Yellow (Y)
1 = Magenta (M)
2 = Cyan (C)
3 = Dye Sublimation Black (K dye)
contrast value (0 thru 10)
error value
successful
failed
ZBRPRNSetContrastIntensityLvl
Description: Sets the color intensity level for the specified image bu
Syntax: int ZBRPRNSetContrastIntensityLvl(
HANDLE hPrinter,
int printerType,
int imgBufIdx,
int intensity,
int *err)
Parameters: hPrinter device context value for a printer driver
printerType printer type value, Appendix B
imgBufIdx image buffer index:
0 = Yellow (Y)
1 = Magenta (M)
2 = Cyan (C)
3 = Dye Sublimation Black (K dye)
intensity intensity value (0 thru 10)
err error value
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)
device context value for a printer driver
printer type value, Appendix B
intensity value (0 thru 10)
error value

Zebra® Card Printer

Software Development Kit
Reference Manual

November 7, 2007.

Card
Printer
Solutions

Copyright Notice

Trademarks

Contents

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
P110i [1]	✓	✓	-	-	-
P120i [2]	✓	✓	-	-	-
P330i [1]	✓	✓	✓	✓	✓
P430i [2]	✓	✓	✓	✓	✓
P630i [3]	-	✓	✓	-	-
P640i [4]	-	✓	✓	-	-

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

ZBRPRNSetContrastIntensity
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 functions 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

ZBRPRNGetSDKVer

Description: Returns the SDK dll version.

Syntax:

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

Parameters:

major	major version number
minor	minor version number
engLevel	engineering level

Printer Driver Handle Functions

ZBRGetHandle

```
Description:  Gets a handle for a printer driver.

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

Parameters:   hPrinter            device context value for a printer driver
              pName               printer driver name
              printerType         printer type value, Appendix B
              err                 error value

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	device context value for a printer driver
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
cmd	command buffer
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
cmd	command buffer
response	response buffer
respSize	number of bytes returned
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
numb	number of times to send the command
cmd	command buffer
err	error value

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	device context value for a printer driver
	printerType	printer type value, Appendix B
	filename	full path of the *.prn file
	err	error value
Return Value:	TRUE	successful
	FALSE	failed

Error Codes: Appendix A

Status Functions

ZBRPRNGetPrintCount

Description: Gets the total number of cards printed.

Syntax:

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

Parameters:

hPrinter	device context value for a printer driver
printerType	printer type value, Appendix B
printCount	total card count
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
serialNumb	serial number buffer
respSize	response size
err	error value

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            device context value for a printer driver
              printerType         printer type value, Appendix B
              options             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            response size
              err                 error value


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          device context value for a printer driver
             printerType       printer type value, Appendix B
             serialNumb        serial number buffer
             respSize          response size
             err               error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
paramIDx	requested parameter (see *Operational Parameters* below)
opParam	operational parameter buffer
respSize	response size
err	error value

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           *errorCode)
```

Parameters:

errorCode	current error code status

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	device context value for a printer driver
printerType	printer type value, Appendix B
err	error value

Return Value:

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

Error Codes: Appendix A

Cleaning Functions

ZBRPRNStartCleaningSeq

```
Description:  Starts a cleaning sequence.

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

Parameters:   hPrinter             device context value for a printer driver
              printerType          printer type value, Appendix B
              err                  error value

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           *cleanCount,
                                  int           *cleanCardCounter,
                                  int           *err)

Parameters:   hPrinter            device context value for a printer driver
              printerType         printer type value, Appendix B
              imgCounter          total number of head-down Image passes (each
                                  ribbon panel used counts as a pass)
              cleanCounter        image passes before a cleaning alert is
                                  sent, default = 5000
              cleanCardCounter    the number of cleaning card passes when
                                  cleaning, default = 5
              err                 error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
ribbonPanelCounter	number of panels printed before start cleaning, default = 5000
cleanCardPass	number of cleaning passes through printer, default = 5
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
checksum	returned checksum
err	error value

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           device context value for a printer driver
              printerType        printer type value, Appendix B
              mode               mode:
                                   0 = printer with card feeder (default)
                                   1 = printer without a card feeder
              err                error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
resistance	print head resistance value in ohms
err	error value

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           device context value for a printer driver
              printerType        printer type value, Appendix B
              err                error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
offset	offset value in dots
err	error value

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            device context value for a printer driver
              printerType         printer type value, Appendix B
              offset              offset value in dots
              err                 error value

Note:         300 dots per inch

Return Value: TRUE                successful
              FALSE               failed

Error Codes:  Appendix A
```

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	device context value for a printer driver
printerType	printer type value, Appendix B
offset	offset value in dots
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
offset	offset value in dots
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
imgBufIdx	image buffer index: 0 = Yellow (Y) 1 = Magenta (M) 2 = Cyan (C) 3 = Dye Sublimation Black (K dye)
contrast	contrast value (0 thru 10)
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
imgBufIdx	image buffer index: 0 = Yellow (Y) 1 = Magenta (M) 2 = Cyan (C) 3 = Dye Sublimation Black (K dye)
intensity	intensity value (0 thru 10)
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
intensity	intensity value (0 thru 10)
err	error value

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

ZBRPRNSetMonoContrast

Description: Sets the monochrome contrast level.

Syntax:

```
int ZBRPRNSetMonoIntensity(
                  HANDLE        hPrinter,
                  int           printerType,
                  int           contrast,
                  int           *err)
```

Parameters:	hPrinter	device context value for a printer driver
	printerType	printer type value, Appendix B
	contrast	contrast value (0 thru 10)
	err	error value
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	device context value for a printer driver
printerType	printer type value, Appendix B
clrVarnish	clear varnish: 1 = clear varnish overlay image buffer 0 = clear k-resin image buffer
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
colorBufIdx	index to the color buffer: 0 = Yellow (Y) 1 = Magenta (M) 2 = Cyan (C) 3 = Dye Sublimation Black (K dye)
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
printParam	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	error value

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            device context value for a printer driver
              printerType         printer type value, Appendix B
              imgBufIdx           color image buffer index:
                                    0 = Yellow (Y)
                                    1 = Magenta (M)
                                    2 = Cyan (C)
                                    3 = Dye Sublimation Black (K dye)
              err                 error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
printParam	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	error value

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             device context value for a printer driver
              printerType          printer type value, Appendix B
              printParam            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                  error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
startX	start X position in dots
startY	start Y position in dots
width	width of the box in dots
height	height of the box in dots
thickness	line thickness in dots
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
startX	start X position in dots
startY	start Y position in dots
width	width of the box in dots
height	height of the box in dots
thickness	line thickness in dots
gMode	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	1 = use varnish overlay
err	error value

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            device context value for a printer driver
              printerType         printer type value, Appendix B
              startX              start X position in dots
              startY              start Y position in dots
              rotation            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              1 = bold
              height              height in dots of the text box:
                                    104 = 28 point normal
                                    140 = 28 point bold
              text                text buffer
              err                 error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
startX	start X position in dots
startY	start Y position in dots
rotation	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	1 = bold
width	width in dots of the text box, if 0 scales according to height
height	height in dots of the text box: 104 = 28 point normal 140 = 28 point bold
gMode	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	text buffer
isVarnish	1 = use varnish overlay
err	error value

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            device context value for a printer driver
              printerType         printer type value, Appendix B
              xWidth              end of print x axis in dots
              err                 error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
err	error value

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            device context value for a printer driver
              printerType         printer type value, Appendix B
              err                 error value

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           device context value for a printer driver
              printerType        printer type value, Appendix B
              err                error value

Return Value: TRUE               successful
              FALSE              failed

Error Codes:  Appendix A
```

ZBRPRNMoveCard

Description: Moves the card a specified distance.

Syntax:

```
int ZBRPRNMoveCardFwd(
                  HANDLE        hPrinter,
                  int           printerType,
                  int           count,
                  int           *err)
```

Parameters:

hPrinter	device context value for a printer driver
printerType	printer type value, Appendix B
count	distance (count 100 = 8 mm / 0.315 in) to move: positive number moves the card forward negative number moves the card backward
err	error value

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            device context value for a printer driver
              printerType         printer type value, Appendix B
              err                 error value

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            device context value for a printer driver
              printerType         printer type value, Appendix B
              cardType            card type:
                                    0 = standard test card
                                    1 = printer test card
                                    2 = magnetic encoder test card
                                    3 = lamination test card
              err                 error value

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	device context value for printer driver
printerType	printer type value, Appendix B
startX	start X position in dots
startY	start Y position in dots
rotation	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	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	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	2 .. 9 (see Appendix D)
barcodeHeight	bar code height in dots (see Appendix D)
textUnder	1 = yes 0 = no
barcodeData	barcode buffer (see Appendix D)
err	error value

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             device context value for a printer driver
              printerType          printer type value, Appendix B
              dir                  direction:
                                     0 = forward
                                     1 = reverse
              err                  error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
trkNumb	track number: 1 = track 1 2 = track 2 3 = track 3
density	encoding density (75 or 210)
err	error value

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            device context value for a printer driver
              printerType         printer type value, Appendix B
              err                 error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
coercivity	coercivity: 0 = low 1 = high
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
std	encoding standard: 0 = JIS 1 = ISO (default)
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
trksToRead	values ORed to determine tracks to read: 0x01 = track 1 0x02 = track 2 0x04 = track 3
trk1Buf	response buffer from track 1
respSizeTrk1	number of bytes returned for track 1
trk2Buf	response buffer for track 2
respSizeTrk2	number of bytes returned from track 2
trk3Buf	response buffer for track 3
respSizeTrk3	number of bytes returned from track 3
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
trkNumb	track number: 1 = track 1 2 = track 2 3 = track 3
trkBuf	response buffer
respSize	number of bytes returned
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
trksToWrite	values ORed to determine tracks to write: 0x01 = track 1 0x02 = track 2 0x04 = track 3
trk1Data	data buffer for track 1
trk2Data	data buffer for track 2
trk3Data	data buffer for track 3
err	error value

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	device context value for a printer driver
printerType	printer type value, Appendix B
trkNumb	track number: 1 = track 1 2 = track 2 3 = track 3
trkData	data buffer
err	error value

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           trksToWrite,
                  char          *trk1Data,
                  char          *trk2Data,
                  char          *trk3Data,
                  int           *err)
```

Parameters:

hDC	handle to the printer’s graphical context
printerType	printer type value, Appendix B
trksToWrite	values ORed to determine tracks to read: 0x01 = track 1 0x02 = track 2 0x04 = track 3
trk1Data	data buffer for track 1
trk2Data	data buffer for track 2
trk3Data	data buffer for track 3
err	error value

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);
```

Printer Error Codes

CODE	ERROR	POSSIBLE CAUSE
-1	ZBR_ERROR_PRINTER_MECHANICAL_ERROR	Mechanical error
0	ZBR_ERROR_NO_ERROR	Indicates that there were no errors
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
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

CODE	ERROR	POSSIBLE CAUSE
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
60	ZBR_ERROR_PRINTER_NOT_SUPPORTED	Printer not supported
61	ZBR_ERROR_CANNOT_GET_PRINTER_HANDLE	Unable to open handle to Zebra printer driver
62	ZBR_ERROR_CANNOT_GET_PRINTER_DRIVER	Cannot open printer driver
63	ZBR_ERROR_GETPRINTERDATA_ERROR	Windows API error -- GetLastError() function of Win32 API will provide with more extended error information
64	ZBR_ERROR_INVALID_MAG_TRK_NUMB	The magnetic track number does not exist (e.g., not in 1... 3 range)
65	ZBR_ERROR_INVALID_PRINTER_HANDLE	Invalid printer handle
66	ZBR_ERROR_CLOSEPRINTER_FAILURE	Error closing printer driver handle

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
- P630i (except barcode functions)
- P640i (except barcode functions)

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

ZBRGDIGetSDKVer

Description: Returns the SDK dll version.

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

Parameters:

major	major version number
minor	minor version number
engLevel	engineering level

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	printer driver name
hDC	device context value
err	error value

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	device context value
err	error value

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	device context value
	err	error value

Return Value:		
	TRUE	successful
	FALSE	failed

Error Codes: Appendix A

ZBRGDIClearGraphics

```
Description:  Clears the graphic buffer.

Syntax:       int ZBRGDIClearGraphics(
                                  int          *err)

Parameters:   err                 error value

Return Value: TRUE                successful
              FALSE               failed

Error Codes:  Appendix A
```

Print Functions

ZBRGDIPrintGraphics

Description: Prints the graphic buffer.

Syntax:

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

Parameters:

hDC	device context value
err	error value

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	device context value
filename	image filename
position	position: 0 = ZBR_UPPER_LEFT 1 = ZBR_LOWER_LEFT 2 = ZBR_UPPER_RIGHT 3 = ZBR_LOWER_RIGHT 4 = ZBR_CENTERED
err	error value

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	device context value
filename	image filename
x	x position of the top-left corner
y	y position of the top-left corner
width	rectangle width in dots
height	rectangle height in dots
err	error value

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	printer driver name
err	returned error value

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

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

Note: 300 dots per inch

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	starting x position for the line in dots
y1	starting y position for the line in dots
x2	ending x position for the line in dots
y2	ending y position for the line in dots
color	RGB value
thickness	thickness in dots
err	error value

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	name of the file that contains the image
x	x position of top-left corner of image
y	y position of top-left corner of image
err	error value

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            name of the file that contains the image
              position            position
                                    0 = upper left
                                    1 = lower left
                                    2 = upper right
                                    3 = lower right
                                    4 = centered
              err                 error value

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	name of the file that contains the image
x	x position of top-left corner of rectangle
y	y position of top-left corner of rectangle
width	rectangle width in dots
height	rectangle height in dots
err	error value

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	x position of top-left corner of rectangle
y	y position of top-left corner of rectangle
width	rectangle width in dots
height	rectangle height in dots
thickness	line thickness for the rectangle
color	RGB color value
err	error value

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	x position of top-left corner the rectangle
y	y position of top-left corner of rectangle
width	width of the ellipse
heigth	height of the ellipse
thickness	line thickness for the ellipse in dots
color	RGB color value
err	error value

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           startX,
                  int           startY,
                  int           rotation,
                  int           barcodeType,
                  int           barWidthRatio,
                  int           barcodeMultiplier,
                  int           barcodeHeight,
                  int           textUnder,
                  char          *barcodeData,
                  int           *err)
```

Parameters:

startX	start X position in dots
startY	start Y position in dots
rotation	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	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	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	barcode multiplier
barcodeHeight	bar code height in dots
textUnder	1 = yes 0 = no
barcodeData	barcode buffer
err	error value

Note: 300 dots per inch

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

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_PD_ERROR_DLG_CANCELED	User closed or canceled the DLG window
8026	ZBR_PD_ERROR_SETUP_FAILURE	PrintDlg function failed to load the required resources
8027	ZBR_PD_ERROR_PARSE_FAILURE	PrintDlg function failed to parse the strings in the [devices] section of the WIN.INI file
8028	ZBR_PD_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_PD_ERROR_LOAD_DRV_FAILURE	PrintDlg function failed to load the device driver for the specified printer
8030	ZBR_PD_ERROR_GET_DEVMODE_FAIL	Printer driver failed to initialize a DEVMODE structure
8031	ZBR_PD_ERROR_INIT_FAILURE	PrintDlg function failed during initialization, and there is no more specific extended error code to describe the failure
8032	ZBR_PD_ERROR_NO_DEVICES	No printer drivers were found
8033	8032 ZBR_PD_ERROR_NO_DEFAULT_PRINTER	A default printer does not exist
8034	ZBR_PD_ERROR_DN_DM_MISMATCH	Data in the DEVMODE and DEVNAMES structures describes two different printers
8035	ZBR_PD_ERROR_CREATE_IC_FAILURE	PrintDlg function failed when it attempted to create an information context
8036	ZBR_PD_ERROR_PRINTER_NOT_FOUND	The [devices] section of the WIN.INI file did not contain an entry for the requested printer
8037	ZBR_PD_ERROR_DEFAULT_DIFFERENT	Error occurs when you store the DEVNAMES structure, and the user changes the default printer by using the Control Panel

ZBRPRNSetContrastIntensity
ZBRPRNSetHologramIntensity

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
- P630i
- P640i

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

ZBRGCGetSDKVer

Description: Returns the SDK version numbers.

Syntax:

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

Parameters:

major	major version number of the SDK library
minor	minor version number of the SDK library
engLevel	engineering level of the SDK library

Printer Functions

ZBRGetHandle

Description: Gets a handle for a printer driver.

Syntax:

```
int ZBRGetHandle(
                    HANDLE          *hPrinter,
                    char            *pName,
                    int             *printerType,
                    int             *err)
```

Parameters:

hPrinter	printer driver device context value
pName	printer driver name
printerType	printer type value, see Appendix B
err	error value

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	printer driver device context value
err	error value

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	printer driver device context value
printerType	printer type value, see Appendix B
err	error value

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	printer driver device context value
printerType	printer type value, see Appendix B
err	error value

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	printer driver device context value
printerType	printer type value, see Appendix B
eject	1 = eject card after encoding
err	error value

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,
                     unsigned char    *atr,
                     int              *atrSize,
                     int              *err)
```

Parameters:

hPrinter	printer driver device context value
printerType	printer type value, see Appendix B
atr	response buffer
atrSize	byte count of the response
err	error value

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,
                      unsigned int     printerType,
                      unsigned char    cfg,
                      unsigned char    *atr,
                      unsigned int     *atrSize,
                      int              *err)
```

Parameters:

hPrinter	printer driver device context value
printerType	printer type value, see appendix B
cfg	card configuration byte; see *Details* below: X0XXX001 X0XXX010 X0XXX100 X0XXX011 X0XXX110 X0XXX111 0000XXXX 0001XXXX 0010XXXX 1111XXXX 00001000 X0XX1XXX 11111XXX
atr	response buffer
dataOutSize	size of the response buffer
dataOutSizeNeeded	byte count of the response
err	error value

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 (maximum 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.

ZBRGCCardPowerDown

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

Syntax:

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

Parameters:

hPrinter	printer driver device context value
printerType	printer type value, see Appendix B
err	error value

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,
                    unsigned char   *dataIn,
                    int             dataInSize,
                    unsigned char   *dataOut,
                    int             dataOutSize,
                    int             *respSize,
                    int             *err)
```

Parameters:

hPrinter	printer driver device context value
printerType	printer type value, see Appendix B
dataIn	APDU buffer
dataInSize	size of the APDU buffer
dataOut	response buffer
dataOutSize	size of response buffer
respSize	byte count of the response
err	error value

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,
                                     unsigned char    *dataIn,
                                     int              dataInSize,
                                     unsigned char    *dataOut,
                                     int              *respSize,
                                     int              *err)

Parameters:   hPrinter               printer driver device context value
              printerType            printer type value, see Appendix B
              dataIn                 APDU buffer
              dataInSize             size of the APDU buffer
              dataOut                response buffer
              respSize               byte count of the response
              err                    error value

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,
                     unsigned         int printerType,
                     char             *statusData,
                     unsigned int     *respSize,
                     int              *err)
```

Parameters:

hPrinter	printer driver device context value
printerType	printer type value, see appendix B
statusData	pointer to buffer where status data is copied
respSize	byte count of the response
err	error value

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	printer driver device context value
printerType	printer type value, see Appendix B
cardType	card type value, see Appendix B
err	error value

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,
                     unsigned char   *dirData,
                     int             *respSize,
                     int             *err)
```

Parameters:

hPrinter	printer driver device context value
printerType	printer type value, see Appendix B
dirData	response buffer
respSize	byte count of the response
err	error value

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,
                     unsigned char    *readerVer,
                     int              *respSize,
                     int              *err)
```

Parameters:

hPrinter	printer driver device context value
printerType	printer type value, see Appendix B
readerVer	response buffer
respSize	byte count of the response
err	error value

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	printer driver device context value
printerType	printer type value, see Appendix B
mode	operating mode: 0 = ISO mode 1 = EMV mode err error value

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               printer driver device context value
              printerType            printer type value, see Appendix B
              mode                   operating mode:
                                       0 = ISO mode
                                       1 = EMV mode
              err                    error value

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,
                                    unsigned char    *timeoutValue,
                                    int              *err)

Parameters:   hPrinter              printer driver device context value
              printerType           printer type value, see Appendix B
              timeoutValue          current timeout value of the reader in
                                    seconds; 0 = infinite
              err                   error value

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,
                      unsigned char    timeoutValue,
                      int              *err)
```

Parameters:

hPrinter	printer driver device context value
printerType	printer type value, see Appendix B
timeoutValue	timeout value of the reader to be set to, 0 = infinite
err	error value

Return Value:

TRUE	successful
FALSE	failed

Error Codes: Appendix A

GemCore Error Codes

CODE	ERROR	POSSIBLE CAUSE
5000	ZBR_ERROR_GETPRINTERDATA_FAILURE	Encoding error
5001	(RESERVED)	-
5002	(RESERVED)	-
5003	ZBR_ERROR_START_CARD_ERROR	Error positioning card and receiving response
5004	ZBR_ERROR_EJECT_CARD_ERROR	Error ejecting card after encoding
5005	ZBR_ERROR_END_CARD_ERROR	Error ending Smart Encoding process
5006	ZBR_ERROR_SMARTCARD_READ_ERROR	Error reading Smart Card Reader
5007	ZBR_ERROR_SMARTCARD_WRITE_ERROR	Error sending data to Reader
5008	ZBR_ERROR_BUFFER_OVERFLOW	Response is to large for buffer
5009	(RESERVED)	-
5010	ZBR_ERROR_RESETTING_SMARTCARD	Error resetting Smart Card
5011	(RESERVED)	-
5012	(RESERVED)	-
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

CODE	ERROR	POSSIBLE CAUSE
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
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
5042	ZBR_ERROR_DATA_TOO_LONG	Response data is larger than response buffer size

CODE	ERROR	POSSIBLE CAUSE
5043	ZBR_ERROR_DATA_TOO_SHORT	Invalid data returned
5044	ZBR_ERROR_DATA_OVERFLOW	Data is larger than the data buffer
5046	ZBR_ERROR_GETDATA_TIMEOUT	Reader time-out error
5047	ZBR_ERROR_BUFFER_TOO_SMALL	Receiving buffer to small for returned data
5048	ZBR_ERROR_CARD_SHORT_CIRCUITING	The card is consuming too much electricity or is short circuiting
5049	ZBR_ERROR_SETPRINTERDATA_FAILURE	Error communicating with printer
5050	ZBR_ERROR_NO_ACK_FROM_PRINTER	No acknowledgement received
5051	ZBR_ERROR_PRINTER_NOT_OK	No response after a send operation
5053	ZBR_ERROR_UNKNOWN_ERROR	Unknown Smart Card Error
5054	ZBR_ERROR_ON_POWER_DOWN	Power-down error
5055	ZBR_ERROR_ON_POWER_UP	Power-up error
5056	ZBR_ERROR_READ_SMARTCARD	Read error
5057	(RESERVED)	-
5058	ZBR_ERROR_INVALID_PRINTER_TYPE	Not a valid Zebra Card Printer
5059	ZBR_ERROR_INVALID_CARD_TYPE	Invalid Smart Card Type
5060	ZBR_ERROR_INVALID_POINTER	Null pointer
5061	ZBR_ERROR_INVALID_WRITE_ADDRESS	Invalid Smart Card Address
5062	ZBR_ERROR_MEMORY_OVERFLOW	Buffer to small for returned data
5063	ZBR_ERROR_SMARTCARD_NOT_SUPPORTED	Smart Card Type not supported
5064	ZBR_ERROR_INVALID_READ_ADDRESS	Invalid Smart Card Address
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

CODE	ERROR	POSSIBLE CAUSE
5068	ZBR_CARD_ALREADY_POWERED_ON	Already powered on
5069	ZBR_ERROR_UNKNOWN_ERROR_CODE	Undefined error

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

ZBRGPMFGetSDKVer

Description: Returns the SDK version numbers.

Syntax:

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

Parameters:

major	major version number
minor	minor version number
engLevel	engineering level number

Printer Functions

ZBRGetHandle

Description: Gets a handle for a printer driver.

Syntax:

```
int ZBRGetHandle(
                  LPHANDLE            hPrinter,
                  LPSTR               pName,
                  int                 *printerType,
                  int                 *err)
```

Parameters:

hPrinter	returned printer driver handle
pName	printer driver name
printerType	returned printer type value, see Appendix B
err	returned error value

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	printer driver handle
err	returned error value

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	printer driver handle
printerType	printer type value, see Appendix B
err	returned error value

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	printer driver handle
printerType	printer type value, see Appendix B
err	returned error value

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	printer driver handle
printerType	printer type value, see Appendix B
eject	eject: 1 = eject the card after encoding 0 = position the card for printing
err	returned error value

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,
                                 unsigned char       blockNumber,
                                 unsigned char       keyAB,
                                 unsigned char       *key,
                                 int                 *err)

Parameter:   hPrinter            printer driver handle
             printerType         printer type value, see Appendix B
             blockNumber         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               defines if key to load is a Key A or a Key B :
                                   0 = KeyA
                                   1 = KeyB
             key                 pointer to 6 bytes key be loaded.
             err                 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,
                  unsigned char      cardType,
                  unsigned char      blockNumber,
                  unsigned char      keyAB,
                  int                *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
cardType	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	block number to authenticate: 0 to 63 with GEMEASY_8000 0 to 255 with GEMCOMBI or GEMEASY_32000
keyAB	defines if the key to load is Key A or Key B: 0 = KeyA 1 = KeyB
err	returned error value

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,
                                  unsigned char       cardType,
                                  unsigned char       blockNumber,
                                  unsigned char       *dataBlock,
                                  unsigned int        *dataBlockSize,
                                  int                 *err)

Parameters:   hPrinter            printer driver handle
              printerType         printer type value, see Appendix B
              cardType            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         block number to authenticate:
                                    0 to 63 with GEMEASY_8000
                                    0 to 255 with GEMCOMBI or GEMEASY_32000
              dataBlock           pointer to the data block read
              dataBlockSize       pointer to read dataBlock buffer size (16)
              err                 returned error value

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,
                  unsigned char     cardType,
                  unsigned char     blockNumber,
                  unsigned char     *dataBlock,
                  unsigned int      *dataBlockSize,
                  int               *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
cardType	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	block number to authenticate: 0 to 63 with GEMEASY_8000 0 to 255 with GEMCOMBI or GEMEASY_32000
dataBlock	pointer to the write data buffer
dataBlockSize	pointer to dataBlock buffer size (16)
err	returned error value

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,
                  unsigned char     cardType,
                  unsigned char     blockNumber,
                  long              value,
                  int               *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
cardType	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	block number to authenticate: 0 to 62 with GEMEASY_8000 0 to 254 with GEMCOMBI or GEMEASY_32000
value	value to be subtracted (-2147483647 to +2147483648)
err	returned error value

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,
                  unsigned char       cardType,
                  unsigned char       blockNumber,
                  long                value,
                  int                 *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
cardType	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	block number to authenticate: 0 to 62 with GEMEASY_8000 0 to 254 with GEMCOMBI or GEMEASY_32000
value	value to be subtracted (-2147483647 to +2147483648)
err	returned error value

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,
                  unsigned char     cardType,
                  unsigned char     blockNumber,
                  int               *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
cardType	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	block number to authenticate: 0 to 62 with GEMEASY_8000 0 to 254 with GEMCOMBI or GEMEASY_32000
err	returned error value

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,
                                  unsigned char       cardType,
                                  unsigned char       blockNumber,
                                  int                 *err)

Parameters:   hPrinter            printer driver handle
              printerType         printer type value, see Appendix B
              cardType            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         block number to authenticate:
                                    0 to 62 with GEMEASY_8000
                                    0 to 254 with GEMCOMBI or GEMEASY_32000
              err                 returned error value

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,
                  unsigned char       cardType,
                  unsigned char       blockNumber,
                  unsigned char       authentication,
                  long                value,
                  int                 *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
cardType	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	block number to authenticate: 0 to 60 with GEMEASY_8000 0 to 124 with GEMCOMBI or GEMEASY_32000
authentication	automatic authentication control: 0 = No Authentication 1 = Authentication KeyA 2 = Authentication KeyB
value	initial purse value to format the block (-2147483647 to + 2147483648)
err	returned error value

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,
                                  unsigned char       cardType,
                                  unsigned char       blockNumber,
                                  unsigned char       authentication,
                                  long                *value,
                                  int                 *err)

Parameters:   hPrinter            printer driver handle
              printerType         printer type value, see Appendix B
              cardType            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         block number to authenticate:
                                    0 to 60 with GEMEASY_8000
                                    0 to 124 with GEMCOMBI or GEMEASY_32000
              authentication      automatic authentication control:
                                    0 = No Authentication
                                    1 = Authentication KeyA
                                    2 = Authentication KeyB
              value               pointer to read purse value
                                    (-2147483647 to + 2147483648)
              err                 returned error value

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,
                  unsigned char       cardType,
                  unsigned char       blockNumber,
                  unsigned char       authentication,
                  long                value,
                  int                 *err)
```

Parameters:

```
hPrinter            printer driver handle
printerType         printer type value, see Appendix B
cardType            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         block number to authenticate:
                      0 to 60 with GEMEASY_8000
                      0 to 124 with GEMCOMBI or GEMEASY_32000
authentication      automatic authentication control:
                      0 = No Authentication
                      1 = Authentication KeyA
                      2 = Authentication KeyB
value               value to debit from the purse
                      (-2147483647 to + 2147483648)
err                 returned error value
```

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,
                  unsigned char      cardType,
                  unsigned char      blockNumber,
                  unsigned char      authentication,
                  long               value,
                  int                *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
cardType	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	block number to authenticate: 0 to 60 with GEMEASY_8000 0 to 124 with GEMCOMBI or GEMEASY_32000
authentication	automatic authentication control: 0 = No Authentication 1 = Authentication KeyA 2 = Authentication KeyB
value	value to add to the purse (-2147483647 to + 2147483648)
err	returned error value

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,
                  unsigned int        tryTime,
                  unsigned int        *data,
                  unsigned int        *dataSize,
                  int                 *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
tryTime	try time duration for reader to get a MAD card 0 to 256 by step of 100ms 0 = one try only
data	pointer to response buffer
dataSize	pointer to size of buffer to store response
err	returned error value

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,
                  unsigned char      cardType,
                  unsigned char      blockNumber,
                  unsigned char      authentication,
                  unsigned char      *data,
                  unsigned int       *dataSize,
                  int                *err)
```

Parameters:

```
hPrinter          printer driver handle
printerType       printer type value, see Appendix B
cardType          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       block number to read from:
                     0 to 62 with GEMEASY_8000
                     0 to 254 with GEMCOMBI or GEMEASY_32000
authentication    automatic authentication control:
                     0 = No Authentication
                     1 = Authentication KeyA
                     2 = Authentication KeyB
data              pointer to the data to read
dataSize          pointer to the data buffer size
                     in  = number of bytes to read
                     out = number of bytes read
err               returned error value
```

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,
                  unsigned char       cardType,
                  unsigned char       blockNumber,
                  unsigned char       authentication,
                  unsigned char       writeVerify,
                  unsigned char       *data,
                  unsigned int        *dataSize,
                  int                 *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
cardType	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	block number to authenticate: 0 to 63 with GEMEASY_8000 0 to 255 with GEMCOMBI or GEMEASY_32000
authentication	automatic authentication control: 0 = No Authentication 1 = Authentication KeyA 2 = Authentication KeyB
writeVerify	perform write verification: 0 = No Write Verification 1 = Perform Write Verification
data	pointer to the data to write
dataSize	pointer to the size of the data buffer; i.e., number of bytes to write
err	returned error value

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,
                  unsigned char     cardType,
                  unsigned char     blockNumber,
                  unsigned char     authentication,
                  unsigned char     writeVerify,
                  long              value,
                  unsigned char     data,
                  int               *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
cardType	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	block number to authenticate: 0 to 62 with GEMEASY_8000 0 to 254 with GEMCOMBI or GEMEASY_32000
authentication	automatic authentication control 0 = No Authentication 1 = Authentication KeyA 2 = Authentication KeyB
writeVerify	perform write verification 0 = No Write Verification 1 = Perform Write Verification
value	initial value to format the block (-2147483647 to + 2147483648)
data	user data byte
err	returned error value

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,
                  unsigned char      cardType,
                  unsigned char      blockNumber,
                  unsigned char      authentication,
                  long               *value,
                  unsigned char      *data,
                  int                *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
cardType	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	block number to authenticate: 0 to 62 with GEMEASY_8000 0 to 254 with GEMCOMBI or GEMEASY_32000
authentication	automatic authentication control: 0 = No Authentication 1 = Authentication KeyA 2 = Authentication KeyB
value	pointer to read value
data	pointer to read user data byte
err	returned error value

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,
                  unsigned char       cardType,
                  unsigned char       blockNumber,
                  unsigned char       authentication,
                  long                value,
                  unsigned char       destinationBlock,
                  int                 *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
cardType	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	source block: 0 to 62 with GEMEASY_8000 0 to 254 with GEMCOMBI or GEMEASY_32000
authentication	automatic authentication control: 0 = No Authentication 1 = Authentication KeyA 2 = Authentication KeyB
value	value to be subtracted from source block (-2147483647 to + 2147483648)
destinationBlock	destination block to store the result: 0 to 30 with GEMEASY_8000 0 to 254 with GEMCOMBI
err	returned error value

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,
                  unsigned char     cardType,
                  unsigned char     blockNumber,
                  unsigned char     authentication,
                  long              value,
                  unsigned char     destinationBlock,
                  int               *err)
```

Parameters:

```
hPrinter          printer driver handle
printerType       printer type value, see Appendix B
cardType          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       source block:
                    0 to 62 with GEMEASY_8000
                    0 to 254 with GEMCOMBI or GEMEASY_32000
authentication    automatic authentication control:
                    0 = No Authentication
                    1 = Authentication KeyA
                    2 = Authentication KeyB
value             value to be added from source block
                    (-2147483647 to + 2147483648)
destinationBlock  destination block to store the result:
                    0 to 30 with GEMEASY_8000
                    0 to 254 with GEMCOMBI
err               returned error value
```

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,
                  unsigned char     cardType,
                  unsigned char     blockNumber,
                  unsigned char     authentication,
                  unsigned char     destinationBlock,
                  int               *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
cardType	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	source block: 0 to 62 with GEMEASY_8000 0 to 254 with GEMCOMBI or GEMEASY_32000
authentication	automatic authentication control: 0 = No Authentication 1 = Authentication KeyA 2 = Authentication KeyB
destinationBlock	destination block to store the result: 0 to 30 with GEMEASY_8000 0 to 254 with GEMCOMBI
err	returned error value

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,
                  unsigned char      cardType,
                  unsigned char      blockNumber,
                  unsigned char      authentication,
                  unsigned char      *keyA,
                  unsigned char      accessBitsB0,
                  unsigned char      accessBitsB1,
                  unsigned char      accessBitsB2,
                  unsigned char      accessBitsB3,
                  unsigned char      *keyB,
                  int                *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
cardType	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	source block: 0 to 63 with GEMEASY_8000, 0 to 255 with GEMCOMBI or GEMEASY_32000.
authentication	automatic authentication control: 0 = No Authentication 1 = Authentication KeyA 2 = Authentication KeyB
keyA	keyA to write in the sector trailer
accessBitsB0	access condition bits B0(0 to 7)
accessBitsB1	access condition bits B1(0 to 7)
accessBitsB2	access condition bits B2(0 to 7)
accessBitsB3	access condition bits B3(0 to 7)
keyB	keyB to write in the sector trailer
err	returned error value

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 191.

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,
                  unsigned char      versionType
                  unsigned char      *firmware
                  int                *len
                  int                *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
versionType	fw version to return: 1 = ROS Version 2 = OROS Version
firmware	pointer to buffer to receive the firmware info
len	character-size of the buffer
err	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,
                                  unsigned char      *productTypeID
                                  int                *productTypeIDSize
                                  unsigned char      *productSN
                                  int                *prodSNLen
                                  int                *err)

Parameters:   hPrinter            printer driver handle
              printerType         printer type value, see Appendix B
              productTypeID       pointer to the product type identifier
              productTypeIDSize   pointer:
                                    [in] pointer to size of productTypeID buffer
                                    [out] pointer to product type ID size
              productSN           pointer to the product serial number
              prodSNLen           pointer:
                                    [in] pointer to size of productSN buffer
                                    [out] pointer to product type SN size
              err                 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,
                  unsigned char      *mode,
                  unsigned char      *gbpAddress,
                  int                *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
mode	pointer to current reader mode
gbpAddress	pointer to current GBP address
err	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,
                  unsigned char      mode,
                  int                *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
mode	Reader operating mode 0x00 = Normal Mode (ISO14443A&B + MIFARE) 0x08 = MAD (ISO14443A + MIFARE) 0x0F = PayPass (ISO14443A&B)
err	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,
                  unsigned char     address,
                  unsigned char     *data,
                  int               *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
address	address to read from - 0 to 15
data	pointer to the value in EEPROM
err	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,
                  unsigned char     address,
                  unsigned char     data,
                  int               *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
address	address to read from - 0 to 15
data	value to write to EEPROM
err	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,
                  unsigned char      *baudRateTypeA,
                  unsigned char      *baudRateTypeB,
                  int                *err)
```

Parameter:

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

Return Value:

TRUE	successful
FALSE	failed, check error codes

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,
                  unsigned char       mode,
                  int                 *err)
```

Parameter:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
mode	pointer controls the RF state: 1 = RF On 2 = RF Off 3 = RF Reset
err	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,
                                  unsigned char      modType,
                                  int                *err)

Parameter:   hPrinter             printer driver handle
             printerType          printer type value, see Appendix B
             modType              pointer controls the RF state:
                                    0 = Type A
                                    1 = Type B
             err                  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,
                  unsigned char     *modType,
                  int               *err)
```

Parameter:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
modType	pointer to returned modulation type: 0 = Type A 1 = Type B
err	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,
                   unsigned char            card,
                   unsigned short           *ATQA,
                   int                      *err)
```

Parameters:

```
hPrinter           printer driver handle
printerType        printer type value, see Appendix B
card               First card or Next card:
                     0 = First Card
                     1 = Next Card
ATQA               pointer to card ATQA
err                returned error value
```

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,
                  unsigned char      cascadeLevel,
                  unsigned char      *cascadeLevelSN,
                  int                cascadeLevelSNLen,
                  int                *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
cascadeLevel	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	pointer to card serial number
cascadeLevelSNLen	size of serial number array (minimum = 4)
err	returned error value

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,
                                  unsigned char      cascadeLevel,
                                  unsigned char      *cascadeLevelSN,
                                  int                cascadeLevelSNLen,
                                  unsigned char      *SAK,
                                  int                *err)

Parameters:   hPrinter            printer driver handle
              printerType         printer type value, see Appendix B
              cascadeLevel        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      pointer to card serial number
              cascadeLevelSNLen   size of serial number array (minimum = 4)
              SAK                 pointer to Select Acknowledge Type A of card
              err                 returned error value

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	printer driver handle
printerType	printer type value, see Appendix B
err	returned error value

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

Combined 14443A Functions

ZBRGPMF_ISO14443_3_A_GetCard

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            printer driver handle
printerType         printer type value, see Appendix B
cardAndTimeout      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
                          unsigned char ucCard;
                          //0 = Timeout Not Specified
                          //1 = Timeout Specified
                          unsigned char ucIsTimeoutSpecified;
                          // Only taken into account in case
                          timeout set as specified unsigned char
                          ucTimeout_50msBased;
                      }sCARD_AND_TIMEOUT;
getCardAInfo        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;
                          unsigned char *pucSerialNumber;
                          unsigned short usATQA;
                          unsigned char ucSAK;
                      }sGET_CARD_A;
err                 returned error value
```

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 189.

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,
                  unsigned char      *serialNumber,
                  int                serialNumberLen,
                  unsigned char      *SAK,
                  int                *err)
```

Parameters:

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

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,
                  unsigned char       PPSBaudRates
                  int                 *err)
```

Parameters:

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

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 189.

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,
                  unsigned char      PPSBaudRates,
                  unsigned char      *serialNumber,
                  int                serialNumberLen,
                  sGET_CARD_A_T_CL   *getCardATclInfo,
                  unsigned char      *SAK,
                  int                *err)
```

Parameters:

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

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 189.

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,
                   unsigned char       CID,
                   unsigned char       *ATS,
                   int                 *ATSSize,
                   int                 *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
CID	Desired card identifier from CID_MIN(0) to CID_MAX(14)
ATS	Pointer to card Answer To Select
ATSSize	Pointer to ATS size: [In]: Size of the ATS buffer, min size = 1 [Out]: ATS length
err	returned error value

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,
                  unsigned char      CID,
                  unsigned char      DSI,
                  unsigned char      DRI,
                  int                *err)
```

Parameters:

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

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,
                  unsigned char      CID,
                  unsigned char      NAD,
                  int                commandLength,
                  unsigned char      *command,
                  unsigned char      *response,
                  int                *responseSize,
                  int                *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
CID	Desired card identifier from CID_MIN(0) to CID_MAX(14)
NAD	Node Address Logical Connection ISO7816-3 compliant, SAD and DAD between 0 and 7
commandLength	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	Pointer to the command to send to the card
response	Pointer to the data returned from the card
responseSize	Pointer to length of data returned by the card [In]: Length of response buffer, min size = 1 [Out]: Length of data returned from the card
err	returned error value

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,
                  unsigned char      CID,
                  int                *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
CID	Desired card identifier from CID_MIN(0) to CID_MAX(14)
err	returned error value

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,
                                 unsigned char      CID,
                                 int                *err)

Parameters:   hPrinter           printer driver handle
              printerType        printer type value, see Appendix B
              CID                Desired card identifier from CID_MIN(0)
                                   to CID_MAX(14)
              err                returned error value

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,
                  unsigned char      *response,
                  int                *responseSize
                  int                *err)
```

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed, check error codes

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,
                  unsigned char      card,
                  unsigned char      AFI,
                  unsigned char      numberOfSlots,
                  sATQB              *ATQB,
                  int                *err)
```

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

Structure Definitions: See “Structure Definitions” on page 189.

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,
                  unsigned char      slotNumber,
                  sATQB              *ATQB,
                  int                *err)
```

Parameters:

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

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

Structure Definitions: See “Structure Definitions” on page 189.

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,
                  unsigned char                   CID,
                  unsigned char                   bitRate,
                  unsigned char                   *maxLenBufInd_CID,
                  int                             *err)
```

Parameters	hPrinter	printer driver handle
	printerType	printer type value, see Appendix B
	PUPIId	pointer to sPseudoUniquePICCIdentifier structure
	protocolInfo	pointer to protocolInfo structure
	CID	desired card identifier: 0 to 14
	bitRate	desired bit rates
	maxLenBufInd_CID	pointer to answer to attribute
	err	returned error value
Return Value:	TRUE	successful
	FALSE	failed, check error codes

Error Codes: Appendix A

Structure Definitions: See “Structure Definitions” on page 189.

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	printer driver handle
printerType	printer type value, see Appendix B
PUPIId	pointer to sPseudoUniquePICCIdentifier structure
err	returned error value

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

Structure Definitions: See “Structure Definitions” on page 189.

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,
                  unsigned char       AFI,
                  unsigned char       bitrates,
                  sGET_CARD_B         *getCardBInfo,
                  int                 *err)
```

Parameters:

hPrinter — printer driver handle

printerType — printer type value, see Appendix B

cardAndTimeout — pointer to structure (see definition) indicating first or next card and whether the timeout is used

AFI — application family identifier

bitRates — mask bit of allowed baudrates to be used by the Attrib, same coding as the Bit_Rate_capability of Protocol Info

getCardBInfo — pointer to structure (see definition) giving Type B card info

err — returned error value

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 189.

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,
                  unsigned char      *dataIn,
                  unsigned int       dataInSize,
                  unsigned char      *dataOut,
                  unsigned int       *dataOutSize,
                  unsigned int       *dataOutSizeNeeded,
                  int                *err)
```

Parameters:

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

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,
                  unsigned long      timeout,
                  int                *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B

timeout	3-byte timeout value - 0x000000 to 0x3FC000
	T = timeout * 128 / 13.56 microsecond
	Max = 39.47 seconds (0x3FC000)

err	returned error value

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

Structure Definitions

```
RF_ON                                               1
RF_OFF                                              2
RF_RESET                                            3

MAIN_CARD_INTERFACE                                 1
AUXILIARY_CARD_INTERFACE                            0

FIRST_CARD                                          0
NEXT_CARD                                           1
FIRST_CARD_T_CL                                     2
NEXT_CARD_T_CL                                      3

TIMEOUT_NOT_SPECIFIED                               0
TIMEOUT_SPECIFIED                                   1

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

STAT_PROTOCOL_MASK_BIT                              0x08
STAT_PROTOCOL_T0                                    0x00
STAT_PROTOCOL_T1                                    0x08

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_847                 0x04


typedef struct S_PSEUDO_UNIQUE_PICC_IDENTIFIER
{
   unsigned char ucByte[4];
}sPseudoUniquePICCIdentifier;


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


typedef struct S_PROTOCOL_INFO
{
   unsigned char ucBitRateCapability;
   unsigned char ucMaxFrameSize_ProtocolType;
   unsigned char ucFrameWaitingTimeInteger_ApplicationDataCoding_FrameOption;
}sProtocolInfo;


typedef struct S_CARD_AND_TIMEOUT
{
   unsigned char ucCard;
   unsigned char ucIsTimeoutSpecified;
   unsigned char ucTimeout_50msBased;
}sCARD_AND_TIMEOUT;
```

```
typedef struct S_ATQB
{
   unsigned char ucFirstByte;
   sPseudoUniquePICCIdentifier sPUPIId;
   sApplicationData sAppData;
   sProtocolInfo sProInfo;
}sATQB;


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 */
   unsigned char *pucSerialNumber;
   unsigned short usATQA;
   unsigned char ucSAK;
}sGET_CARD_A;


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


typedef struct S_GET_CARD_B
{
   sATQB sCurATQB;
   unsigned char ucATTRIB;
   unsigned char 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		

MIFARE Error Codes

CODE	ERROR	POSSIBLE CAUSE
0	ZBR_ERROR_NO_ERROR	Indicates that there were no errors
7001	ZBR_ERROR_INVALID_PRINTER_TYPE	Invalid printer type
7002	ZBR_ERROR_INVALID_POINTER	Invalid pointer
7003	ZBR_ERROR_START_CARD_ERROR	Error positioning card and receiving response
7010	ZBR_ERROR_NO_ACK_FROM_PRINTER	No acknowledgment from printer
7012	ZBR_ERROR_BUFFER_TOO_SMALL	Receiving buffer too small for returned data
7013	ZBR_ERROR_UNKNOWN_ERROR	Unknown error
7014	ZBR_ERROR_WRONG_BUFFER_SIZE	Wrong buffer size
7017	ZBR_ERROR_RECEIVED_NO_DATA	No data received
7018	ZBR_ MIFARE_ERROR_PARAMETERS_ERROR	Wrong number of parameters or a value is incorrect
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

CODE	ERROR	POSSIBLE CAUSE
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
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

CODE	ERROR	POSSIBLE CAUSE
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
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

CODE	ERROR	POSSIBLE CAUSE
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

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

- P330i
- P430i

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

```
Description:  Returns the SDK version numbers.

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

Parameters:   major                     major version number
              minor                     minor version number
              patchLevel                engineering level number
```

ZBRGetHandle

```
Description:  Gets a handle for a printer driver.

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

Parameters:   hPrinter                  returned printer driver handle
              pName                     printer driver name
              printerType               returned printer type value, see Appendix B
              err                       returned error value

Return Value: TRUE                      successful
              FALSE                     failed, check error codes

Error Codes:  Appendix A
```

ZBRCloseHandle

Description: Closes the printer driver handle.

Syntax:

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

Parameters:

hPrinter	printer driver handle
err	returned error value

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,
                    DWORD           printerType,
                    int             *err)
```

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
err	error value

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: Ejects the card.

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

Parameters:  hPrinter           printer driver handle
             printerType        printer type value, see Appendix B
             err                returned error value

Note:        all this function after communication with the reader is finished
             and before calling ZBRCloseHandle.

Return Value:TRUE               successful
             FALSE              failed, check error codes

Error Codes: Appendix A
```

ZBRUHFEndCardEx

Description: Ejects the card.

Syntax:

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

Parameters:

hPrinter	printer driver handle
printerType	printer type value, see Appendix B
eject	1 = eject card 0 = position card for printing
err	error value

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

Return Value:

TRUE	successfull
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRUHFSend

Description: Sends the given data to the UHF reader.

Syntax:

```
INT ZBRUHFSend(
                    HANDLE          hPrinter,
                    DWORD           printerType,
                    LPBYTE          dataIn,
                    DWORD           dataInSize,
                    int*            err)
```

Parameters:

hPrinter	printer handle
printerType	printer type value, see appendix B
dataIn	input buffer
dataInSize	input buffer size
err	returned error value

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

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRUHFReceive

Description: Receives data from the UHF reader.

Syntax:

```
INT ZBRUHFReceive(
                    HANDLE          hPrinter,
                    DWORD           printerType,
                    LPBYTE          dataOut,
                    DWORD           dataOutSize,
                    LPDWORD         dataOutSizeNeeded,
                    int*            err)
```

Parameters:

hPrinter	printer handle
printerType	printer type
dataOut	output buffer where data will be received
dataOutSize	output buffer size
dataOutSizeNeeded	output buffer size needed
err	returned error value

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

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRUHFWriteTagData

Description: Writes to the data section of the tag.

Syntax:

```
INT ZBRUHFWriteTagDataEx(
                    HANDLE          hPrinter,
                    DWORD           printerType,
                    BYTE            memBank,
                    DWORD           addr,
                    LPBYTE          dataIn,
                    BYTE            dataInSize,
                    int*            err)
```

Parameters:

hPrinter	printer handle
printerType	printer type

memBank	Gen2 cards:
	0x00(Reserved)
	0x01(EPC)
	0x02(TID)
	0x03(User)

addr	offset from the memBank origin
dataIn	pointer to data buffer
dataInSize	number of bytes to be written
err	returned error value

Return Value:

TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

ZBRUHFReadTagData

Description: Reads data from the tag.

Syntax:

```
INT ZBRUHFReadTagData(
                    HANDLE          hPrinter,
                    DWORD           printerType,
                    BYTE            memBank,
                    DWORD           addr,
                    BYTE            wordCount
                    LPBYTE          dataOut,
                    DWORD           dataOutSize,
                    LPDWORD         dataOutSizeNeeded,
                    int*            err)
```

Parameters:

hPrinter	printer handle
printerType	printer type

memBank	Gen2 cards:
	0x00(Reserved)
	0x01(EPC)
	0x02(TID)
	0x03(User)

addr	offset from the memBank origin
wordCount	number of words to read
dataOut	pointer to read data buffer
dataOutSize	size of dataOut buffer
dataOutSizeNeeded	number of bytes returned
err	returned error value

Return Value:

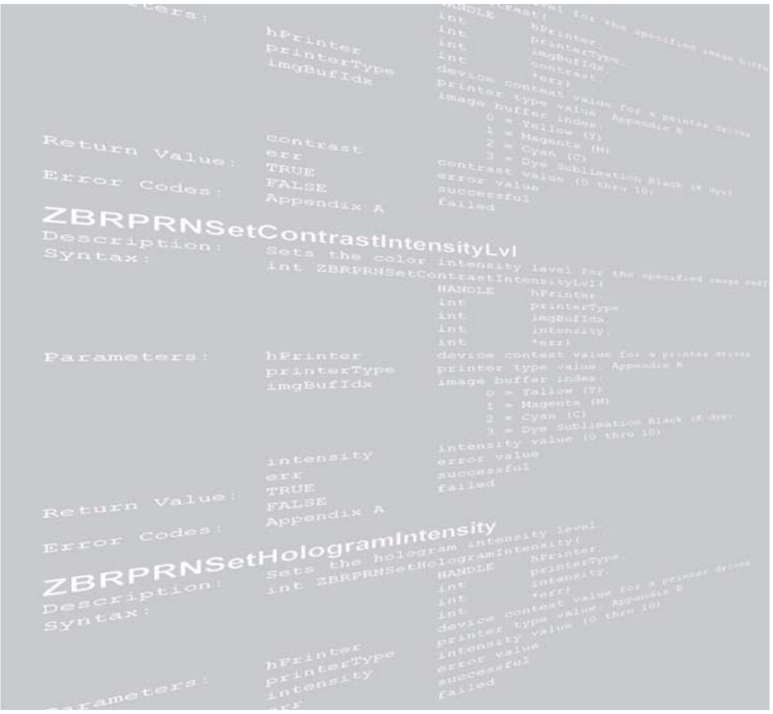
TRUE	successful
FALSE	failed, check error codes

Error Codes: Appendix A

UHF Error Codes

CODE	ERROR	POSSIBLE CAUSE
0	ZBR_ERROR_NO_ERROR	Indicates that there were no errors
9001	ZBR_ERROR_INVALID_PRINTER_TYPE	Invalid printer type
9002	ZBR_ERROR_INVALID_POINTER	Invalid pointer
9003	ZBR_ERROR_NO_ACK_FROM_PRINTER	No acknowledgment from printer
9004	ZBR_ERROR_BUFFER_TOO_SMALL	Receiving buffer too small for returned data
9005	ZBR_ERROR_WRONG_BUFFER_SIZE	Wrong buffer size
9006	ZBR_ERROR_RECEIVED_NO_DATA	No data received
9007	ZBR_ERROR_BUFFEROVERFLOW	Response too large for buffer
9008	ZBR_ERROR_INVALID_BAUD_RATE	Invalid baud rate
9009	ZBR_ERROR_INVALID_DATA_SIZE	Invalid data size
9010	ZBR_ERROR_UNKNOWN_ERROR	Unknown error (internal)
9040	ZBR_UHF_ERROR_GENERAL_TAG_ERROR	Error occurred during read, write, lock, or kill command
9041	ZBR_UHF_ERROR_DATA_TOO_LARGE	Data value is larger than expected or is not the correct size
9042	ZBR_UHF_ERROR_PROTOCOL_INVALID_KILL_PASSWORD	Wrong password included in kill command
9100	ZBR_UHF_ERROR_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

CODE	ERROR	POSSIBLE CAUSE
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
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
9602	ZBR_UHF_ERROR_TAG_ID_BUFFER_REPEATED_ TAG_ID	Tag ID in Tag ID Buffer is duplicated
9603	ZBR_UHF_ERROR_TAG_ID_BUFFER_NUM_TAG_TOO_ LARGE	Number of tags exceeds the maximum number of supported tags


Return Value:
Error Codes:
ZBRPRNSetContrastIntensityLvl
Description:
Syntax:
Parameters:
hPrinter
printerType
imgBufIdx
intensity
Return Value:
Error Codes:
ZBRPRNSetHologramIntensity
Description:
Syntax:

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

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, unsigned int prnType,
                                            int *errValue);
funcMovePrintReady movePrintReady;

       // Magnetic encoder functions

typedef int (CALLBACK *funcReadMag)(HANDLE prnHandle, unsigned 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, unsigned 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,
                                  "B5018780618000001541^John Doe ^4912101”,    // track 1
                                  "5018780618000001541=4912101678”,            // track 2
                                  "0000000000000000001241”,                    // 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,
                                       unsigned char *dataIn, int dataInSize,
                                       unsigned char *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;
       unsigned char   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

       unsigned char 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

       unsigned char   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;
       unsigned char   addr       = 32;
       unsigned char   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};
       unsigned char   dataIn[10];
       for (int i=0; i<dataSize; i++) dataIn[i] = 0x41 + i;

// Continued on next page
```

```
        unsigned char 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

        unsigned char   rdCode[]   = {  0x16,0x00,0xB0,0x00,addr,0x00,(unsigned
                                        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
        unsigned char 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
        unsigned char 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;
        }


        unsigned char 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
        unsigned char authentication = 0x01; // 0 = No Authentication, 1 = KeyA, 2 = KeyB
        unsigned char keyAB = 0x00;          // 0 = Key A, 1 = Key B
        unsigned char blockNumber = 0x04;    // Selected Block Number
        unsigned char writeVerify = 0x01;    // 0 = No Write Verify, 1 = Write Verify

        unsigned char* keyDataA = new unsigned char[6];    // Key to load (6 byte key value)
        unsigned int dataSize = 48;                        // Write data size
        unsigned char* data = new unsigned char[dataSize]; // Write data array (3 blocks)
        unsigned int outDataSize = 16;                     // Read data size (1 block)
        unsigned char* dataOut = new unsigned char[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 (unsigned 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.

```
// C++ ZBRUHFReader.dll Example
// ****************************************************************************************
// ****************************************************************************************

bool loadZBRUHFFunctions()
{
       // Load the Zebra UHF SDK library
       HMODULE hModule = LoadLibrary("ZBRUHFReader.dll");
       if (hModule)
       {
              printf("ZBRUHFReader.dll has been successfully loaded.\n");
       }
       else
       {
              printf("Error loading ZBRUHFReader.dll.\n");
              return FALSE;
       }

       getHandle = (ZBRGetHandle)GetProcAddress(hModule, "ZBRGetHandle");
       closeHandle = (ZBRCloseHandle)GetProcAddress(hModule, "ZBRCloseHandle");

       getSDKVer = (ZBRUHFGetSDKVer)GetProcAddress(hModule, "ZBRUHFGetSDKVer");

       startCard = (ZBRUHFStartCard)GetProcAddress(hModule, "ZBRUHFStartCard");
       endCard = (ZBRUHFEndCard)GetProcAddress(hModule, "ZBRUHFEndCard");
       endCardEx = (ZBRUHFEndCardEx)GetProcAddress(hModule, "ZBRUHFEndCardEx");

       sendData = (ZBRUHFSend)GetProcAddress(hModule, "ZBRUHFSend");
       receiveData = (ZBRUHFReceive)GetProcAddress(hModule, "ZBRUHFReceive");

       readTagData = (ZBRUHFReadTagData)GetProcAddress(hModule, "ZBRUHFReadTagData");
       writeTagData = (ZBRUHFWriteTagData)GetProcAddress(hModule, "ZBRUHFWriteTagData");

       return true;
}


int main(int argc, char* argv[])
{
       if (!loadZBRUHFFunctions())
       {
              return 0;
       }

       HANDLE hPrinter = NULL;
       int printerType = NULL;
       int err = 0,
              ret = 0;

       // ZBRGetHandle
       ret = getHandle(&hPrinter, "Zebra P330i USB Card Printer", &printerType, &err);
```

// Continued on next page

```
// ZBRUHFGetSDKVer
int major,
       minor,
       engLevel;

getSDKVer(&major, &minor, &engLevel);
printf("\nZBRUHFGetSDKVer: %d.%d.%d", major, minor, engLevel);


// ZBRUHFSend
BYTE writeCmd[3]; // Get Reader Information
writeCmd[0] = 0xFF;
writeCmd[1] = 0x00;
writeCmd[2] = 0x03;

if (!sendData(hPrinter, printerType, writeCmd, 3, &err))
{
       printf("ZBRUHFSend Error: %d", err);
}
else
{
       printf("\n\nZBRUHFSend: Command %02x%02x%02x (Get Reader Info) Succeeded",
                writeCmd[0], writeCmd[1], writeCmd[2]);

       // ZBRUHFReceive
       LPBYTE dataOut = new BYTE[1024];
       DWORD dataOutSizeNeeded = 0;

       if (!receiveData(hPrinter, printerType, dataOut, 1024, &dataOutSizeNeeded,
                &err))
       {
                printf("\nZBRUHFReceive Error: %d", err);
       }
       else
       {
                printf("\nZBRUHFReceive: ");

                printf("\n\n\tBootLoader Ver: ");
                for (int i = 5; i < 9; i++)
                {
                           printf("%02x", dataOut[i]);
                }

                printf("\n\tHardware Ver: ");
                for (i = 9; i < 13; i++)
                {
                           printf("%02x", dataOut[i]);
                }

                printf("\n\tFirmware Date: ");
                for (i = 13; i < 17; i++)
                {
                           printf("%02x", dataOut[i]);
                }

                printf("\n\tFirmware Ver: ");
                for (i = 17; i < 21; i++)
                {
                           printf("%02x", dataOut[i]);
                }
```

// Continued on next page

```
                                printf("\n\tSupported Protocols: ");
                                for (i = 21; i < 25; i++)
                                {
                                        printf("%02x", dataOut[i]);
                                }
                        }
                        // cleanup
                        if (dataOut)
                        {
                                delete [] dataOut;
                                dataOut = NULL;
                        }
                }


                // ZBRUHFStartCard
                if (!startCard(hPrinter, printerType, &err))
                {
                        printf("\nZBRUHFStartCard Error: %d", err);
                }
                else
                {
                        printf("\nZBRUHFStartCard Succeeded");
                }


                // ZBRUHFWriteTagData
                BYTE membank = 0x01; // EPC
                DWORD address = 0x00000002;

                BYTE dataSize = 5;
                LPBYTE dataIn = new BYTE[dataSize];
                dataIn[0] = 0xaa; // byte 1
                dataIn[1] = 0xBB; // byte 2
                dataIn[2] = 0x12; // byte 3
                dataIn[3] = 0x34; // byte 4
                dataIn[4] = 0x56; // byte 5

                if (!writeTagData(hPrinter, printerType, membank, address, dataIn, dataSize, &err))
                {
                        printf("\n\nZBRUHFWriteTagData Error: %d", err);
                }
                else
                {
                        printf("\n\nZBRUHFWriteTagData: ");
                        for (unsigned int i = 0; i < dataSize; i++)
                        {
                                printf("%02x ", dataIn[i]);
                        }
                        printf("Succeeded");
                }
                // cleanup
                if (dataIn)
                {
                        delete [] dataIn;
                        dataIn = NULL;
                }
```

// Continued on next page

```
        // ZBRUHFReadTagData
        BYTE wordCount = dataSize;
        DWORD dataOutSizeNeeded = 0;
        LPBYTE dataOut = new BYTE[1024];

        if (!readTagData(hPrinter, printerType, membank, address, wordCount, dataOut, 1024,
               &dataOutSizeNeeded, &err))
        {
               printf("\n\nZBRUHFReadTagData Error: %d", err);
        }
        else
        {
               printf("\nZBRUHFReadTagData: ");
               for (unsigned int i = 0; i < dataOutSizeNeeded; i++)
               {
                        printf("%02x ", dataOut[i]);
               }
        }
        // cleanup
        if (dataOut)
        {
               delete [] dataOut;
               dataOut = NULL;
        }


        // ZBRUHFEndCard
        int eject = 1;
        if (!endCardEx(hPrinter, printerType, eject, &err))
        {
               printf("\n\nZBRUHFEndCard Error: %d", err);
        }
        else
        {
               printf("\n\nZBRUHFEndCard Succeeded");
        }

        ret = closeHandle(hPrinter, &err);

        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.

Printer Error Codes

CODE	ERROR	POSSIBLE CAUSE
-1	ZBR_ERROR_PRINTER_MECHANICAL_ERROR	Mechanical error
0	ZBR_ERROR_NO_ERROR	Indicates that there were no errors
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

CODE	ERROR	POSSIBLE CAUSE
33	ZBR_ERROR_GRAPHIC_IMAGE_DATA_CHECKSUM_ERROR	Graphic data checksum error
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
60	ZBR_ERROR_PRINTER_NOT_SUPPORTED	Printer not supported
61	ZBR_ERROR_CANNOT_GET_PRINTER_HANDLE	Unable to open handle to Zebra printer driver
62	ZBR_ERROR_CANNOT_GET_PRINTER_DRIVER	Cannot open printer driver

CODE	ERROR	POSSIBLE CAUSE
63	ZBR_ERROR_GETPRINTERDATA_ERROR	Windows API error -- GetLastError() function of Win32 API will provide with more extended error information
64	ZBR_ERROR_INVALID_MAG_TRK_NUMB	The magnetic track number does not exist (e.g., not in 1... 3 range)
65	ZBR_ERROR_INVALID_PRINTER_HANDLE	Invalid printer handle
66	ZBR_ERROR_CLOSEPRINTER_FAILURE	Error closing printer driver handle

GemCore Error Codes

CODE	ERROR	POSSIBLE CAUSE
5000	ZBR_ERROR_GETPRINTERDATA_FAILURE	Encoding error
5001	(RESERVED)	-
5002	(RESERVED)	-
5003	ZBR_ERROR_START_CARD_ERROR	Error positioning card and receiving response
5004	ZBR_ERROR_EJECT_CARD_ERROR	Error ejecting card after encoding
5005	ZBR_ERROR_END_CARD_ERROR	Error ending Smart Encoding process
5006	ZBR_ERROR_SMARTCARD_READ_ERROR	Error reading Smart Card Reader
5007	ZBR_ERROR_SMARTCARD_WRITE_ERROR	Error sending data to Reader
5008	ZBR_ERROR_BUFFER_OVERFLOW	Response is to large for buffer
5009	(RESERVED)	-
5010	ZBR_ERROR_RESETTING_SMARTCARD	Error resetting Smart Card
5011	(RESERVED)	-
5012	(RESERVED)	-
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

CODE	ERROR	POSSIBLE CAUSE
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
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
5042	ZBR_ERROR_DATA_TOO_LONG	Response data is larger than response buffer size

CODE	ERROR	POSSIBLE CAUSE
5043	ZBR_ERROR_DATA_TOO_SHORT	Invalid data returned
5044	ZBR_ERROR_DATA_OVERFLOW	Data is larger than the data buffer
5046	ZBR_ERROR_GETDATA_TIMEOUT	Reader time-out error
5047	ZBR_ERROR_BUFFER_TOO_SMALL	Receiving buffer to small for returned data
5048	ZBR_ERROR_CARD_SHORT_CIRCUITING	The card is consuming too much electricity or is short circuiting
5049	ZBR_ERROR_SETPRINTERDATA_FAILURE	Error communicating with printer
5050	ZBR_ERROR_NO_ACK_FROM_PRINTER	No acknowledgement received
5051	ZBR_ERROR_PRINTER_NOT_OK	No response after a send operation
5053	ZBR_ERROR_UNKNOWN_ERROR	Unknown Smart Card Error
5054	ZBR_ERROR_ON_POWER_DOWN	Power-down error
5055	ZBR_ERROR_ON_POWER_UP	Power-up error
5056	ZBR_ERROR_READ_SMARTCARD	Read error
5057	(RESERVED)	-
5058	ZBR_ERROR_INVALID_PRINTER_TYPE	Not a valid Zebra Card Printer
5059	ZBR_ERROR_INVALID_CARD_TYPE	Invalid Smart Card Type
5060	ZBR_ERROR_INVALID_POINTER	Null pointer
5061	ZBR_ERROR_INVALID_WRITE_ADDRESS	Invalid Smart Card Address
5062	ZBR_ERROR_MEMORY_OVERFLOW	Buffer to small for returned data
5063	ZBR_ERROR_SMARTCARD_NOT_SUPPORTED	Smart Card Type not supported
5064	ZBR_ERROR_INVALID_READ_ADDRESS	Invalid Smart Card Address
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

CODE	ERROR	POSSIBLE CAUSE
5068	ZBR_CARD_ALREADY_POWERED_ON	Already powered on
5069	ZBR_ERROR_UNKNOWN_ERROR_CODE	Undefined error

MIFARE Error Codes

CODE	ERROR	POSSIBLE CAUSE
0	ZBR_ERROR_NO_ERROR	Indicates that there were no errors
7001	ZBR_ERROR_INVALID_PRINTER_TYPE	Invalid printer type
7002	ZBR_ERROR_INVALID_POINTER	Invalid pointer
7003	ZBR_ERROR_START_CARD_ERROR	Error positioning card and receiving response
7010	ZBR_ERROR_NO_ACK_FROM_PRINTER	No acknowledgment from printer
7012	ZBR_ERROR_BUFFER_TOO_SMALL	Receiving buffer too small for returned data
7013	ZBR_ERROR_UNKNOWN_ERROR	Unknown error
7014	ZBR_ERROR_WRONG_BUFFER_SIZE	Wrong buffer size
7017	ZBR_ERROR_RECEIVED_NO_DATA	No data received
7018	ZBR_ MIFARE_ERROR_PARAMETERS_ERROR	Wrong number of parameters or a value is incorrect
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

CODE	ERROR	POSSIBLE CAUSE
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
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

CODE	ERROR	POSSIBLE CAUSE
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
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

CODE	ERROR	POSSIBLE CAUSE
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
8023	ZBR_GDI_ERROR_GRAPHICS_NOT_INITIALIZED	Graphics buffer has not been initialized

CODE	ERROR	POSSIBLE CAUSE
8024	ZBR_GDI_ERROR_GETTING_DEVICE_CONTEXT	Unable to create the device context for the driver
8025	ZBR_PD_ERROR_DLG_CANCELED	User closed or canceled the DLG window
8026	ZBR_PD_ERROR_SETUP_FAILURE	PrintDlg function failed to load the required resources
8027	ZBR_PD_ERROR_PARSE_FAILURE	PrintDlg function failed to parse the strings in the [devices] section of the WIN.INI file
8028	ZBR_PD_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_PD_ERROR_LOAD_DRV_FAILURE	PrintDlg function failed to load the device driver for the specified printer
8030	ZBR_PD_ERROR_GET_DEVMODE_FAIL	Printer driver failed to initialize a DEVMODE structure
8031	ZBR_PD_ERROR_INIT_FAILURE	PrintDlg function failed during initialization, and there is no more specific extended error code to describe the failure
8032	ZBR_PD_ERROR_NO_DEVICES	No printer drivers were found
8033	8032 ZBR_PD_ERROR_NO_DEFAULT_PRINTER	A default printer does not exist
8034	ZBR_PD_ERROR_DN_DM_MISMATCH	Data in the DEVMODE and DEVNAMES structures describes two different printers
8035	ZBR_PD_ERROR_CREATE_IC_FAILURE	PrintDlg function failed when it attempted to create an information context
8036	ZBR_PD_ERROR_PRINTER_NOT_FOUND	The [devices] section of the WIN.INI file did not contain an entry for the requested printer
8037	ZBR_PD_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
0	ZBR_ERROR_NO_ERROR	Indicates that there were no errors
9001	ZBR_ERROR_INVALID_PRINTER_TYPE	Invalid printer type
9002	ZBR_ERROR_INVALID_POINTER	Invalid pointer
9003	ZBR_ERROR_NO_ACK_FROM_PRINTER	No acknowledgment from printer
9004	ZBR_ERROR_BUFFER_TOO_SMALL	Receiving buffer too small for returned data
9005	ZBR_ERROR_WRONG_BUFFER_SIZE	Wrong buffer size
9006	ZBR_ERROR_RECEIVED_NO_DATA	No data received
9007	ZBR_ERROR_BUFFEROVERFLOW	Response too large for buffer
9008	ZBR_ERROR_INVALID_BAUD_RATE	Invalid baud rate
9009	ZBR_ERROR_INVALID_DATA_SIZE	Invalid data size
9010	ZBR_ERROR_UNKNOWN_ERROR	Unknown error (internal)
9040	ZBR_UHF_ERROR_GENERAL_TAG_ERROR	Error occurred during read, write, lock, or kill command
9041	ZBR_UHF_ERROR_DATA_TOO_LARGE	Data value is larger than expected or is not the correct size
9042	ZBR_UHF_ERROR_PROTOCOL_INVALID_KILL_PASSWORD	Wrong password included in kill command
9100	ZBR_UHF_ERROR_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

CODE	ERROR	POSSIBLE CAUSE
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
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
9602	ZBR_UHF_ERROR_TAG_ID_BUFFER_REPEATED_TAG_ID	Tag ID in Tag ID Buffer is duplicated
9603	ZBR_UHF_ERROR_TAG_ID_BUFFER_NUM_TAG_TOO_LARGE	Number of tags exceeds the maximum number of supported tags

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
P630i	=	630
P640i	=	640

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.

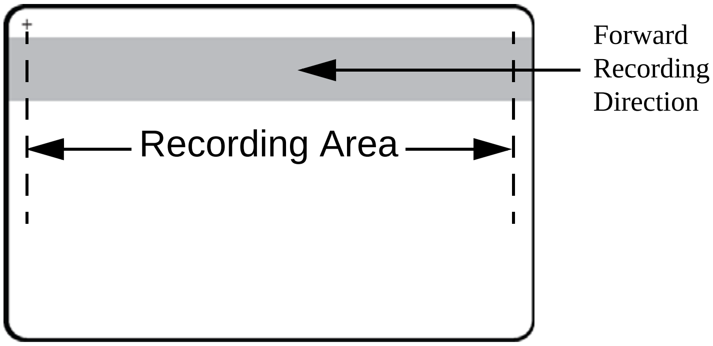
Magnetic Encoders

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.

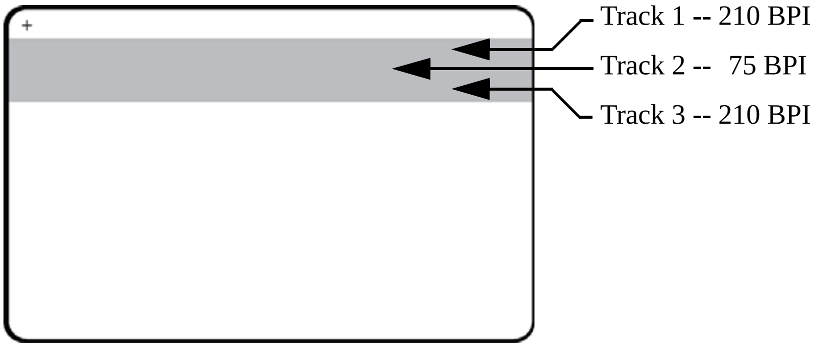
Note that for ISO encoding, the encoder attaches the start, stop, and LRC characters, which should not be included in data downloads.

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.

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)

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
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
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.

ZBRPRNSetContrastIntensity
ZBRPRNSetHologramIntensity

Appendix E

Worldwide Support

For Technical Support or Repair Services, contact the appropriate facility listed below.

North America - Technical Support

Zebra Technologies Card Printer Solutions
1001 Flynn Road
Camarillo, CA 93012-8706 USA

Phone:	+1 800 511 9909
email:	techsupport@zebra.com

North 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 the 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

Europe, Middle East, and Africa - Technical Support

Zebra Technologies Card Printer Solutions
Zebra House, Unit 14, The Valley Centre
Gordon Road, High Wycombe
Buckinghamshire HP13 6EQ, UK

Phone:	+ 44 (0) 8702 411527
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

Latin America - Technical Support

Zebra Technologies Card Printer Solutions
9800 NW 41st Street, Suite 220
Doral, FL 33178

Phone:	+ 1 305 558 3100, extension 2821
e-mail:	techsupport@zebra.com

Latin America - Repair Services

(Please contact North America Repair Services.)

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

ZBRPRNSetContrastIntensity
ZBRPRNSetHologramIntensity

Appendix F

Function Index