# 1. General

**A set of functions for managing all Datecs ESC/POS printers.**

### 1.1. PrnOpen

Opens the serial port and performs initalization of the printer

**Syntax**

int __stdcall PrnOpen(int port, int speed, bool hardware);

**Parameters**

*port*

com port number

*speed*

com port speed

*hardware*

if true, turns on RTS/CTS handshaking, false goes back to XON/OFF

**Return value**

ERR_OK - printer is ready
ERR_TIMEOUT - communication error

### 1.2. PrnClose

Closes the serial port

**Syntax**

int __stdcall PrnClose();

**Return value**

ERR_OK - printer is ready
ERR_TIMEOUT - communication error

### 1.3. PrnSensors

Returns the value of the printer sensors

**Syntax**

int __stdcall PrnSensors();

**Return value**

Value >=0 gives sensor bits
ERR_TIMEOUT - communication error

### 1.4. PrnText

Prints text with specified font/styles.

This function can act as both simple plain text printing and quite complex printing using internal tags to format the text. The function uses the currently font size and style (or default ones) as well as the aligning, however it allows modifications of them inside the text. Any modification of the settings using the tags will be reverted when function completes execution. For example if you have default font selected before using

printText and set bold font inside, it will be reverted to plain when function completes. The tags are control commands used to modify the text printing parameters. They are surrounded by {} brackets. A list of all control tags follows:

* {==} - reverts all settings to their defaults. It includes font size, style, intensity, aligning
* {=Fx} - selects font size. x ranges from 0 to 7 as follows:
*  0: FONT_9X16
*  1: FONT_18X16
*  2: FONT_9X32
*  3: FONT_18X32
*  4: FONT_12X24
*  5: FONT_24X24
*  6: FONT_12X48
*  7: FONT_24X48
* {=L} - left text aligning
* {=C} - center text aligning
* {=R} - right text aligning
* {=J} - justified text
* {=Rx} - text rotation as follows:
*  0: not rotated
*  1: rotated 90 degrees
*  2: rotated 180 degrees
* {=Ix} - sets intensity level as follows:
*  0: intensity 70%
*  1: intensity 80%
*  2: intensity 90%
*  3: intensity 100%
*  4: intensity 120%
*  5: intensity 150%
* {+/-B} - sets or unsets bold font style
* {+/-I} - sets or unsets italic font style
* {+/-U} - sets or unsets underline font style
* {+/-V} - sets or unsets inverse font style
* {+/-W} - sets or unsets text word-wrapping

An example of using tags "{=C}Plain centered text\n{=L}Left centered\n{+B}...bold...{-B}{+I}or ITALIC"

**Syntax**

```
int __stdcall PrnText(char *printtext);
```

**Parameters**

*printtext*

pointer to string containing text to be printed

**Return value**

ERR_OK - operation completed

ERR_TIMEOUT - communication error

### 1.5. PrnImage

Prints image file using specified aligning

**Syntax**

int __stdcall PrnImage(char *file, int align);

**Parameters**

*file*

pointer to string containing the file path and file name of the image file. Supported formats are BMP, JPG, GIF, TIFF, PNG

*align*

bitmap aligning, one of:

| Value | Meaning |
|---|---|
| ALIGN_LEFT(0) | left aligning |
| ALIGN_CENTER(1) | center aligning |
| ALIGN_RIGHT(2) | right aligning |

**Return value**

ERR_OK - operation completed
ERR_TIMEOUT - communication error

### 1.6. PrnLogo

Prints the logo, stored into the printer (if any)

**Syntax**

int __stdcall PrnLogo();

**Return value**

ERR_OK - operation completed
ERR_TIMEOUT - communication error

### 1.7. SetLogo

Sets (or clears) the printer's logo. The logo is persistent.

**Syntax**

int __stdcall PrnSetLogo(char *file);

**Parameters**

*file*

pointer to string containing the file path and file name of the image file. Pass NULL to clear the logo. Supported formats are BMP, JPG, GIF, TIFF, PNG. Every printer have limitations of the supported logo sizes, you can get the sizes with **PrnGetPrinterInfo** function

**Return value**

ERR_OK - operation completed
ERR_TIMEOUT - communication error
ERR_INVALID_PARAMS - either logo file is invalid, cannot be loaded or the logo exceeds the maximum size

### 1.8. PrnIntensity

Changes printing intensity level

**Syntax**

int __stdcall PrnIntensity(int intensity);

**Parameters**

*intensity*

intensity level, one of:

| Value | Meaning |
|---|---|
| INTENSITY_70(0) | 70% |
| INTENSITY_80(1) | 80% |
| INTENSITY_90(2) | 90% |
| INTENSITY_100(3) | 100% |
| INTENSITY_120(4) | 120% |
| INTENSITY_150(5) | 150% |

**Return value**

ERR_OK - operation completed
ERR_TIMEOUT - communication error

### 1.9. PrnPaperCut

Uses the printer cutter if the current printer supports it, or it feeds the paper otherwise

**Syntax**

int __stdcall PrnPaperCut();

**Return value**

ERR_OK - operation completed
ERR_TIMEOUT - communication error

### 1.10. PrnPaperFeed

Feeds the paper X lines

**Syntax**

int __stdcall PrnPaperFeed(int lines);

**Parameters**

*lines*

lines to feed the paper in pixels (1/203 of the inch)

**Return value**

ERR_OK - operation completed
ERR_TIMEOUT - communication error

### 1.11. PrnBarcodeSettings

Sets the various barcode parameters

**Syntax**

int __stdcall PrnBarcodeSettings(int align, int scale, int textpos, int height);

**Parameters**

*align*

barcode aligning, one of:

| Value | Meaning |
|---|---|
| ALIGN_LEFT(0) | left aligning |
| ALIGN_CENTER(1) | center aligning |
| ALIGN_RIGHT(2) | right aligning |

*scale*

barcode scaling, one of:

| Value | Meaning |
|---|---|
| BAR_SCALE_2(2) | 2 pixel lines |
| BAR_SCALE_3(3) | 3 pixel lines |
| BAR_SCALE_4(4) | 4 pixel lines |

*textpos*

the position of the HRI code, one of:

| Value | Meaning |
|---|---|
| BAR_TEXT_NONE(0) | no HRI code is being drawn |
| BAR_TEXT_ABOVE(1) | HRI code drawn above the barcode |
| BAR_TEXT_BELOW(2) | HRI code drawn below the barcode |
| BAR_TEXT_BOTH(3) | HRI code drawn both above and below the barcode |

*height*

barcode height, between 1 and 255, default is 162

**Return value**

ERR_OK - operation completed

ERR_TIMEOUT - communication error

### 1.12. PrnBarcode

Prints a barcode with the parameters specified with **PrnBarcodeSettings**

**Syntax**

int _stdcall PrnBarcode(int type, char *data, int len);

**Parameters**

*type*

barcode type, one of:

| Value | Meaning |
|---|---|
| BAR_UPCA(0) | UPC-A |
| BAR_UPCE(1) | UPC-E |
| BAR_EAN13(2) | EAN-13 |
| BAR_EAN8(3) | EAN-8 |

| | |
|---|---|
| BAR_CODE39(4) | Code 39 |
| BAR_ITF(5) | ITF |
| BAR_CODABAR(6) | Codabar |
| BAR_CODE93(7) | Code 93 |
| BAR_CODE128(8) | Code 128 |
| BAR_PDF417(9) | PDF-417 |

*data*
pointer to barcode data

*len*
the size of the barcode data

**Return value**
ERR_OK - operation completed
ERR_INVALID_PARAMS - some of the input parameters contain wrong value
ERR_UNSUPPORTED - barcode type is not supported
ERR_TIMEOUT - communication error

### 1.13. PrnBarcodePDF417

Prints PDF-417 barcode

**Syntax**

int __stdcall PrnBarcodePDF417(int type, int encoding, int eccl, int size, unsigned char *data, int datalen);

**Parameters**

*type*
PDF-417 type, one of:

| Value | Meaning |
|---|---|
| 0 | Standard |
| 1 | Truncated |

*encoding*
encoding type, one of:

| Value | Meaning |
|---|---|
| 0 | Automatic |
| 1 | Binary |

*eccl*
Error correction control level. Possible values 0 to 9. ECCL=9 automatically selects correction level dependent on data length.

*size*
barcode size, one of:

| Value | Meaning |
|---|---|
| 0 | Width=2, Height=4 |
| 1 | Width=2, Height=9 |
| 2 | Width=2, Height=15 |
| 3 | Width=2, Height=20 |

| | |
|---|---|
| 4 | Width=7, Height=4 |
| 5 | Width=7, Height=9 |
| 6 | Width=7, Height=15 |
| 7 | Width=7, Height=20 |
| 8 | Width=12, Height=4 |
| 9 | Width=12, Height=9 |
| 10 | Width=12, Height=15 |
| 11 | Width=12, Height=20 |
| 12 | Width=20, Height=4 |
| 13 | Width=20, Height=9 |
| 14 | Width=20, Height=15 |
| 15 | Width=20, Height=20 |

*data*
pointer to barcode data

*datalen*
the size of the barcode data

**Return value**
ERR_OK - operation completed
ERR_INVALID_PARAMS - some of the input parameters contain wrong value
ERR_UNSUPPORTED - barcode type is not supported
ERR_TIMEOUT - communication error

### 1.14. PrnBarcodeQRCode

Prints QR CODE barcode

**Syntax**
int __stdcall PrnBarcodeQRCode(int size, int eccl, unsigned char *data, int datalen);

**Parameters**

*size*
barcode symbol size. Possible values: 1, 4, 6, 8, 10, 12, 14

*eccl*
Error correction control level, one of:

| Value | Meaning |
|---|---|
| 1 | 7% |
| 2 | 15% |
| 3 | 25% |
| 4 | 30% |

*data*
pointer to barcode data

*datalen*
the size of the barcode data

**Return value**
ERR_OK - operation completed

ERR_INVALID_PARAMS - some of the input parameters contain wrong value
ERR_UNSUPPORTED - barcode type is not supported
ERR_TIMEOUT - communication error

### 1.15. PrnCodepage

Changes active code page if possible
Some printers require manually enabling this with hardware switch (look for ESC t in the printer's manual)

**Syntax**

int __stdcall PrnCodepage(int codepage);

**Parameters**

*codepage*
code page identifier
OEM code pages:

| Value | Meaning |
|---|---|
| 437 | IBM PC |
| 737 | Greek |
| 775 | Estonian, Lithuanian and Latvian |
| 850 | "Multilingual (Latin-1)" (Western European languages) |
| 852 | "Slavic (Latin-2)" (Central and Eastern European languages) |
| 856 | Cyrillic |
| 857 | Turkish |
| 860 | Portuguese |
| 862 | Hebrew |
| 866 | Cyrillic |

Windows ANSI code pages

| Value | Meaning |
|---|---|
| 1250 | Central and East European Latin |
| 1251 | Cyrillic |
| 1252 | West European Latin |
| 1253 | Greek |
| 1254 | Turkish |
| 1255 | Hebrew |
| 1257 | Baltic |

**Return value**

ERR_OK - operation completed
ERR_TIMEOUT - communication error
ERR_UNSUPPORTED - the printer does not support the codepage selected

### 1.16. PrnPaperWidth

Returns current printer's paper width in pixels (1/203 of inch)

**Syntax**

int __stdcall PrnPaperWidth();

**Return value**

Paper width in pixels

### 1.17. PrnDelimiter

Draws a line with a specified font and char. The line occupies the whole paper width so it is useful for delimiters

**Syntax**

int __stdcall PrnDelimiter(char delimchar, int font);

**Parameters**

*delimchar*

the symbol to be drawn, usually '-'

*font*

font style, one of the:

| Value | Meaning |
|---|---|
| FONT_9X16(0) | Font 9x16 |
| FONT_18X16(1) | Font 18x16 |
| FONT_9X32(2) | Font 9x32 |
| FONT_18X32(3) | Font 18x32 |
| FONT_12X24(4) | Font 12x24 (default) |
| FONT_24X24(5) | Font 24x24 |
| FONT_12X48(6) | Font 12x48 |
| FONT_24X48(7) | Font 24x48 |

**Return value**

ERR_OK - operation completed
ERR_TIMEOUT - communication error

### 1.18. PrnGetCharWidth

Calculates the complete character width (plus intercharacter space) of a single character at given font

**Syntax**

int __stdcall PrnGetCharWidth(int font);

**Parameters**

*font*

font style, one of the:

| Value | Meaning |
|---|---|
| FONT_9X16(0) | Font 9x16 |
| FONT_18X16(1) | Font 18x16 |
| FONT_9X32(2) | Font 9x32 |
| FONT_18X32(3) | Font 18x32 |
| FONT_12X24(4) | Font 12x24 (default) |

| | |
|---|---|
| FONT_24X24(5) | Font 24x24 |
| FONT_12X48(6) | Font 12x48 |
| FONT_24X48(7) | Font 24x48 |

**Return value**

Character width in pixels

### 1.19. PrnGetCharHeight

Calculates the complete character height of a single character at given font

**Syntax**

int __stdcall PrnGetCharHeight(int font);

**Parameters**

*font*

font style, one of the:

| Value | Meaning |
|---|---|
| FONT_9X16(0) | Font 9x16 |
| FONT_18X16(1) | Font 18x16 |
| FONT_9X32(2) | Font 9x32 |
| FONT_18X32(3) | Font 18x32 |
| FONT_12X24(4) | Font 12x24 (default) |
| FONT_24X24(5) | Font 24x24 |
| FONT_12X48(6) | Font 12x48 |
| FONT_24X48(7) | Font 24x48 |

**Return value**

Character height in pixels

### 1.20. PrnGetTextWidth

Calculates the complete text width (plus intercharacter space) at given font

**Syntax**

int __stdcall PrnGetTextWidth(char *text, int font);

**Parameters**

*text*

null-terminated string to calculate the width of

*font*

font style, one of the:

| Value | Meaning |
|---|---|
| FONT_9X16(0) | Font 9x16 |
| FONT_18X16(1) | Font 18x16 |
| FONT_9X32(2) | Font 9x32 |
| FONT_18X32(3) | Font 18x32 |
| FONT_12X24(4) | Font 12x24 (default) |
| FONT_24X24(5) | Font 24x24 |

| | |
|---|---|
| FONT_12X48(6) | Font 12x48 |
| FONT_24X48(7) | Font 24x48 |

**Return value**

Text width in pixels

### 1.21. PrnGetPrinterModel

Returns the currently connected printer model

**Syntax**

int __stdcall PrnGetPrinterModel();

**Parameters**

*text*

null-terminated string to calculate the width of

*font*

font style, one of the:

| Value | Meaning |
|---|---|
| PRINTER_PP50(0) | PP-50 |
| PRINTER_PP55(1) | PP-55 |
| PRINTER_PP60(2) | PP-60 |
| PRINTER_CMP10(3) | CMP-10 |
| PRINTER_DPP250(4) | DPP-250 |
| PRINTER_DPP350(5) | DPP-350 |
| PRINTER_EP50(6) | EP-50 |
| PRINTER_EP55(7) | EP-55 |
| PRINTER_EP60(8) | EP-60 |
| PRINTER_EP300(9) | EP-300 |
| PRINTER_EP1000(10) | EP-1000 |
| PRINTER_DK2300(11) | DK2300 |

**Return value**

Printer model

### 1.22. PrnGetPrinterInfo

Returns combined information about the current printer - name, version, flags, supported modules, etc

**Syntax**

int __stdcall PrnGetPrinterInfo(PRINTER_INFORMATION *information);

**Parameters**

*info*

ponter to PRINTER_INFORMATION structure

**Return value**

ERR_OK upon success

# 2. Magnetic Card

**Functions related to the magnetic stripe reader module.**

### 2.1. MSReadCard

Reads magnetic stripe card

**Syntax**

int MSReadCard(char *track1, char *track2, char *track3, long timeout);

**Parameters**

*track1*

buffer for holding returned card data, minimum of 80 bytes. Pass NULL if you don't want track 1 contents.

*track2*

buffer for holding returned card data, minimum of 41 bytes. Pass NULL if you don't want track 2 contents.

*track3*

buffer for holding returned card data, minimum of 108 bytes. Pass NULL if you don't want track 3 contents.

*timeout*

timeout in miliseconds to read the card data. The actuall scan time may differ, but will be as close as possible to this value

**Return value**

ERR_OK - operation completed
ERR_TIMEOUT - timeout reading card
ERR_UNSUPPORTED - the printer connected does not support reading magnetic cards

### 2.2. MSProcessFinancialCard

Extracts finanial data (such as name, number, expiration date) from financial magnetic card

**Syntax**

int MSProcessFinancialCard(char *track1, char *track2, FINANCIAL_CARD *data);

**Parameters**

*track1*

track 1 data returned from @PrnMSReadCard or NULL

*track2*

track 2 data returned from @PrnMSReadCard or NULL

*data*

pointer to FINANCIAL_CARD structure. Every field that can be decoded will be filled.

**Return value**

ERR_OK - operation completed
ERR_UNSUPPORTED - the card data passed is not from a valid financial card

# 3. SmartCard

**Functions related to the smartcard reader module.**

### 3.1. SCOpen

Initializes and powers up smartcard reader module

**Syntax**

int SCOpen();

**Return value**

ERR_OK - operation completed
ERR_UNSUPPORTED - no smartcard reader module present

### 3.2. SCClose

Powers down smartcard reader module

**Syntax**

int SCClose();

**Return value**

ERR_OK - operation completed
ERR_TIMEOUT - error communicating with the reader

### 3.3. SCClose

Resets the smartcard and returns the ATR. Call this function before sending any APDU commands

**Syntax**

int SCReset(int *protocol, unsigned char *atr, int *atr_length);

**Parameters**

*protocol*

card communication protocol, one of:

| Value | Meaning |
|---|---|
| 0 | T0 |
| 1 | T1 |

*atr*

data buffer where the ATR is returned

*atr_length*

pass the size of atr buffer, upon exit it will contain the count of the bytes written in the atr buffer. If atr buffer is not big enough, then function will return ERR_BUFFER_TOO_SMALL and atr_length will contain the bytes needed

**Return value**

ERR_OK - operation completed
ERR_TIMEOUT - error communicating with the reader
ERR_BUFFER_TOO_SMALL - the size of atr buffer is not enough

### 3.4. SCAPDU_write

Sends APDU write request

**Syntax**

int SCAPDU_write(int cla, int ins, int p1, int p2, unsigned char *write_buffer, int write_length, unsigned char *st1, unsigned char *st2);

**Parameters**

*cla*

the CLA parameter. Consult smartcard documentation.

*ins*

the INS parameter. Consult smartcard documentation.

*p1*

the P1 parameter. Consult smartcard documentation.

*p2*

the P2 parameter. Consult smartcard documentation.

*write_buffer*

pointer to a byte array to write to the card or NULL if you don't want to send any data with the command

*write_length*

the length of the data, pointed by write_buffer

*st1*

pointer to a byte, where ST1 is returned after successfull APDU operation. Pass NULL if you don't want this value.

*st2*

pointer to a byte, where ST2 is returned after successfull APDU operation. Pass NULL if you don't want this value.

**Return value**

ERR_OK - operation completed

ERR_TIMEOUT - error communicating with the reader

### 3.5. SCAPDU_read

Sends APDU read request

**Syntax**

int SCAPDU_read(int cla, int ins, int p1, int p2, unsigned char *read_buffer, int read_length, unsigned char *st1, unsigned char *st2);

**Parameters**

*cla*

the CLA parameter. Consult smartcard documentation.

*ins*

the INS parameter. Consult smartcard documentation.

*p1*

the P1 parameter. Consult smartcard documentation.

*p2*
the P2 parameter. Consult smartcard documentation.
*read_buffer*
pointer to a byte array to write to store the returned card information
*read_length*
the length of the data (0-255 bytes) you want to read
*st1*
pointer to a byte, where ST1 is returned after successfull APDU operation. Pass NULL if you don't want this value.
*st2*
pointer to a byte, where ST2 is returned after successfull APDU operation. Pass NULL if you don't want this value.

**Return value**

Result value greater or equal to zero represents the bytes of data returned from the card, negative value means error.
ERR_OK - operation completed
ERR_TIMEOUT - error communicating with the reader

### 3.6. SCAPDU_combined

Sends combined APDU read/write request

**Syntax**

int SCAPDU_combined(int cla, int ins, int p1, int p2, unsigned char *write_buffer, int write_length, unsigned char *read_buffer, int read_length, unsigned char *st1, unsigned char *st2);

**Parameters**

*cla*
the CLA parameter. Consult smartcard documentation.
*ins*
the INS parameter. Consult smartcard documentation.
*p1*
the P1 parameter. Consult smartcard documentation.
*p2*
the P2 parameter. Consult smartcard documentation.
*write_buffer*
pointer to a byte array to write to the card or NULL if you don't want to send any data with the command
*write_length*
the length of the data, pointed by write_buffer
*read_buffer*
pointer to a byte array to write to store the returned card information
*read_length*
the length of the data (0-255 bytes) you want to read
*st1*
pointer to a byte, where ST1 is returned after successfull APDU operation. Pass NULL if you don't want this value.

*st2*

pointer to a byte, where ST2 is returned after successfull APDU operation. Pass NULL if you don't want this value.

**Return value**

Result value greater or equal to zero represents the bytes of data returned from the card, negative value means error.

ERR_OK - operation completed

ERR_TIMEOUT - error communicating with the reader

# 4. Mifare

**Functions related to the mifare reader module.**

### 4.1. MFIdent

Returns mifare reader identification string. You have to call **MFInit** first

**Syntax**

int MFIdent(char *ident);

**Parameters**

*ident*

returns mifare reader identification string (max 32 characters)

**Return value**

ERR_OK - operation completed
MF_STAT_TIMEOUT - mifare reader failed to respond

### 4.2. MFInit

Initializes and powers on the mifare reader module. Call this function before any other mifare functions.

**Syntax**

int MFInit();

**Return value**

ERR_OK - operation completed
MF_STAT_TIMEOUT - mifare reader failed to respond

### 4.3. MFClose

Powers down mifare reader module. Call this function after you are done with the mifare reader.

**Syntax**

int MFClose();

**Return value**

ERR_OK - operation completed
MF_STAT_TIMEOUT - mifare reader failed to respond

### 4.4. MFRequestCard

Scans for mifare cards in the area

**Syntax**

int MFRequestCard(int type, unsigned long *serial);

**Parameters**

*type*

card communication protocol, one of:

| Value | Meaning |
|---|---|

| | |
|---|---|
| MF_REQUEST_IDLE_CARDS(0) | Return only cards that are not halted |
| MF_REQUEST_ALL_CARDS(1) | Return all cards |

*serial*

upon success, card serial will be returned here

**Return value**

ERR_OK - operation completed, or one of the MF_STAT_* constants upon error

### 4.5. MFSelectCard

Selects a card to operate with

**Syntax**

int MFSelectCard(unsigned long serial);

**Parameters**

*serial*

card serial number, returned from **MFRequestCard**

**Return value**

ERR_OK - operation completed, or one of the MF_STAT_* constants upon error

### 4.6. MFHaltCard

Puts the selected with **MFSelectCard** card into halted state, such card will not show from **MFRequestCard** with type set to MF_REQUEST_IDLE_CARDS

**Syntax**

int MFHaltCard();

**Return value**

ERR_OK - operation completed, or one of the MF_STAT_* constants upon error

### 4.7. MFAuthByKey

Authenticate with mifare card

**Syntax**

int MFAuthByKey(char type, int block, unsigned char key[6]);

**Parameters**

*type*

key type, either 'A' or 'B'

*block*

block number

*key*

6 bytes key

**Return value**

ERR_OK - operation completed, or one of the MF_STAT_* constants upon error

### 4.8. MFRead

Reads a 16 byte block of data

**Syntax**

int MFRead(int address, unsigned char data[16]);

**Parameters**

*address*

address of what to read

*data*

buffer to return the 16 bytes of data if successful

**Return value**

ERR_OK - operation completed, or one of the MF_STAT_* constants upon error

### 4.9. MFWrite

Writes a 16 byte block of data

**Syntax**

int MFWrite(int address, unsigned char data[16]);

**Parameters**

*address*

address to write to

*data*

16 bytes block of data

**Return value**

ERR_OK - operation completed, or one of the MF_STAT_* constants upon error

### 4.10. MFValueOperation

Performs increment/decrement/restore operations on value cards

**Syntax**

int MFValueOperation(int operation, int src_block, int dst_block, long value);

**Parameters**

*operation*

operation to perform, one of:

| Value | Meaning |
|---|---|
| MF_OPERATION_INCREMENT(0xC0) | increment with value |
| MF_OPERATION_DECREMENT(0xC1) | decrement with value |
| MF_OPERATION_RESTORE(0xC2) | restore the value |

*src_block*

source block number

*dst_block*

destination block number

*value*

the value to increment/decrement with

**Return value**

ERR_OK - operation completed, or one of the MF_STAT_* constants upon error

### 4.11. MFRGetSerial

Returns mifare reader's serial number

**Syntax**

```
int MFRGetSerial(long *serial);
```

**Parameters**

*serial*

where the 4 byte serial number will be returned

**Return value**

ERR_OK - operation completed, or one of the MF_STAT_* constants upon error

### 4.12. MFWriteValue

Writes a 4 byte value in the card

**Syntax**

```
int MFWriteValue(int address, long value);
```

**Parameters**

*address*

address to write to

*value*

4 bytes block of data to write

**Return value**

ERR_OK - operation completed, or one of the MF_STAT_* constants upon error

# 5. Barcode

**Functions related to the barcode reader module.**

### 5.1. BarcodeType2Text

Helper function to return string name of barcode type

**Syntax**

```
void BarcodeType2Text(int type, char *text);
```

**Parameters**

*type*

barcode type returned from **BarcodeScan**

*text*

string, where the barcode type is supported. Make sure it is at least 32 bytes long

### 5.2. BarcodeScan

Scans barcode using the built-in barcode scanning engine

**Syntax**

```
int BarcodeScan(int *type, char *barcode, int maxlen);
```

**Parameters**

*type*

upon success, barcode type is returned here, one of the BARCODE_* constants

*barcode*

string, where the barcode data is returned.

*maxlen*

the size of the buffere pointed by barcode parameter

**Return value**

ERR_OK - barcode successfully scanned
ERR_UNSUPPORTED - barcode reader not supported, or barcode engine error
ERR_TIMEOUT - timeout scanning barcode

### 5.3. PrnSetEuroCode

Sets the ASCII code that will be used to output euro sign.

**Syntax**

```
int __stdcall PrnSetEuroCode(int code);
```

**Parameters**

*code*

ASCII code for euro sign (0-255). Passing 0 disables the euro sign. Default is disabled.

**Return value**

ERR_OK - operation completed
ERR_UNSUPPORTED - this printer does not support defining custom euro code
ERR_TIMEOUT - timeout sending the data

ERR_INVALID_PARAMS - code parameter have wrong value

# 6. Direct Communication

**Functions for directly controlling the printer and modules, attached to it**

### 6.1. CommWrite

Direct sending of data to some of the printer's communication channels.
Due to the implemented caching, the data to CHANNEL_PRN is not sent immediatelly, but rather cached untill read function (like @ComRead) is called or it is flushed by @FlushCache. The other channels are not cached.

**Syntax**

int __stdcall CommWrite(int channel, void *data, int size);

**Parameters**

*channel*

communication channel, one of:

| Value | Meaning |
|---|---|
| CHANNEL_PRN(1) | printing channel, all printers support it. It is always open |
| CHANNEL_SMARTCARD(2) | smartcard channel, only some printers support and only in protocol mode. You have to open the channel with @CommOpenChanel first |
| CHANNEL_GPRS(5) | GSM/GPRS channel, only some printers support and only in protocol mode. You have to open the channel with @CommOpenChanel first |
| CHANNEL_UR(16) | Universal Reader channel, only some printers support and only in protocol mode. You have to open the channel with @CommOpenChanel first |

*data*

data to send

*size*

length of the data buffer

**Return value**

ERR_OK - data was successfully sent
ERR_UNSUPPORTED - channel not supported on the current printer/configuration

### 6.2. CommRead

Direct data reading from some of the printer's communication channels. CommRead forces flushing of the write cache before performing the read operation.

**Syntax**

int __stdcall CommRead(int channel, void *data, int size, unsigned long timeout, int stopbyte);

**Parameters**

*channel*

communication channel, one of:

| Value | Meaning |
|---|---|
| CHANNEL_PRN(1) | printing channel, all printers support it. It is always open |
| CHANNEL_SMARTCARD(2) | smartcard channel, only some printers support and only in protocol mode. You have to open the channel with @CommOpenChanel first |

| | |
|---|---|
| CHANNEL_GPRS(5) | GSM/GPRS channel, only some printers support and only in protocol mode. You have to open the channel with @CommOpenChanel first |
| CHANNEL_UR(16) | Universal Reader channel, only some printers support and only in protocol mode. You have to open the channel with @CommOpenChanel first |

*data*

data to store the result

*size*

length of the data buffer

*timeout*

max timeout in milliseconds

*stopbyte*

pass -1 to disable, or any other value [0-255] to force reading to stop when received

**Return value**

Positive value - numbers of bytes read

ERR_UNSUPPORTED - channel not supported on the current printer/configuration

### 6.3. CommOpenChannel

Opens channel for use with **CommRead** and **CommWrite**. CHANNEL_PRN is opened by default

**Syntax**

int __stdcall CommOpenChannel(int channel);

**Parameters**

*channel*

communication channel, one of:

| Value | Meaning |
|---|---|
| CHANNEL_PRN(1) | printing channel, all printers support it. It is always open |
| CHANNEL_SMARTCARD(2) | smartcard channel, only some printers support and only in protocol mode. You have to open the channel with @CommOpenChanel first |
| CHANNEL_GPRS(5) | GSM/GPRS channel, only some printers support and only in protocol mode. You have to open the channel with @CommOpenChanel first |
| CHANNEL_UR(16) | Universal Reader channel, only some printers support and only in protocol mode. You have to open the channel with @CommOpenChanel first |

**Return value**

ERR_OK - data was successfully sent

ERR_UNSUPPORTED - channel not supported on the current printer/configuration

### 6.4. CommCloseChannel

Closes channel previously opened with **CommOpenChannel**. Do not use for CHANNEL_PRN.

**Syntax**

int __stdcall CommCloseChannel(int channel);

**Parameters**

*channel*

communication channel, one of:

| Value | Meaning |
|---|---|
| CHANNEL_PRN(1) | printing channel, all printers support it. It is always open |
| CHANNEL_SMARTCARD(2) | smartcard channel, only some printers support and only in protocol mode. You have to open the channel with @CommOpenChanel first |
| CHANNEL_GPRS(5) | GSM/GPRS channel, only some printers support and only in protocol mode. You have to open the channel with @CommOpenChanel first |
| CHANNEL_UR(16) | Universal Reader channel, only some printers support and only in protocol mode. You have to open the channel with @CommOpenChanel first |

**Return value**

ERR_OK - data was successfully sent

ERR_UNSUPPORTED - channel not supported on the current printer/configuration

### 6.5. CommCommand

Helper function to execute "command" on some channel. The said command is sent as 4 bytes header and data:

[cmd] [00] [insize(hibyte)] [insize(lobyte)] indata

A reply is received with the format

[st1] [st2] [outsize(hibyte)] [outsize(lobyte)] indata

**Syntax**

int __stdcall CommCommand(int channel, int cmd, void *indata, int insize, unsigned char *st1, unsigned char *st2, void *outdata, unsigned long timeout);

**Parameters**

*channel*

communication channel, one of:

| Value | Meaning |
|---|---|
| CHANNEL_PRN(1) | printing channel, all printers support it. It is always open |
| CHANNEL_SMARTCARD(2) | smartcard channel, only some printers support and only in protocol mode. You have to open the channel with @CommOpenChanel first |
| CHANNEL_GPRS(5) | GSM/GPRS channel, only some printers support and only in protocol mode. You have to open the channel with @CommOpenChanel first |
| CHANNEL_UR(16) | Universal Reader channel, only some printers support and only in protocol mode. You have to open the channel with @CommOpenChanel first |

*indata*

data to send with the command

*insize*

length of the data pointed by indata

*st1*

contains status code 1 upon successful execution. Pass NULL if you don't want it

*st2*

contains status code 2 upon successful execution. Pass NULL if you don't want it

*outdata*

buffer, where the result data is stored. Make sure the buffer is big enought to hold the data packet

**Return value**

ERR_OK - data was successfully sent

ERR_UNSUPPORTED - channel not supported on the current printer/configuration