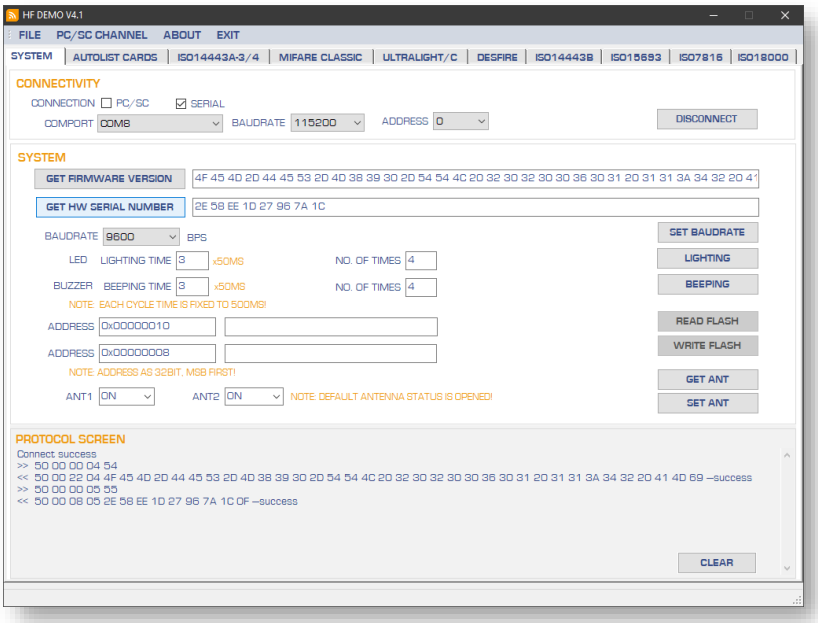


OEM-DESFire Series
13.56 MHz OEM RFID Module
Communication Protocol

iDTRONIC GmbH
Ludwig-Reichling-Straße 4
67059 Ludwigshafen
Germany/Deutschland

Phone: +49 621 6690094-0
Fax: +49 621 6690094-9
E-Mail: info@idtronic.de
Web: idtronic.de

Issue 4.62
– 19. October 2021 –

Contents

1 Description

The communication between the host controller and the reader obeys to a protocol named PARA. This protocol encapsulates the useful data of a message in an invariant frame structure and defines a dialog structure of messages exchanges.

Frame structure

Data is exchanged between the host controller and the reader in blocks, each made up of binary characters on one byte:

4 bytes	0 to 506 bytes	1 byte
Header characters	Data	XOR
	Information field	Checksum

4 bytes header byte includes:*

1st byte								2nd byte	3rd byte	4th byte
A	1	A	1	0	0	0	0			
								Data length to be transmitted excluding header and XOR		Command byte

Remark: A = 0, ACKnowledge of the frame (1st byte = 50)
A = 1, NACK of the frame (message with a status error, 1st byte = F0)

XOR byte: is such that the exclusive-oring of all bytes including XOR is null.

1.1 General dialog structure

The host controller is the master for the transmission; each command from the master is followed by an answer from the reader including the same command byte as the input command.
However, in some cases (card insertion or extraction, time out detection on Rx line or an automatic emergency deactivation of the card) the reader is able to initiate an exchange.

1.1.1 Successful command

System to Reader

50	XX XX	YY	Nnnnnnnnnnnnnnnnnnn	ZZ
ACK	Length	CMD	Data	XOR

Reader to System:

50	UU UU	YY	mmmmmmmmmmmmmmmmmmmm	ZZ
ACK	Length	CMD	Data	XOR

The same command byte YY is returned in the answer from the reader.

1.1.2 Unsuccessful command

System to Reader

50	XX XX	YY	nnnnnnnnnnnnnnnnnnn	ZZ
ACK	Length	CMD	Data	XOR

Reader to System

F0	UU UU	YY	SS	TT
ACK	Length	CMD	Status	XOR

In that case, the status contains the error code information (see error list).

1.1.3 Answer with an acknowledge: (power_off, idle_mode, power_down_mode)

System to Reader (example: PiccHalt)

50	00 00	14	44
ACK	Length	CMD	XOR

Reader to System:

50	00 00	14	44
ACK	Length	CMD	XOR

In the case where the answer is an acknowledge of the command, the reader sends back a frame with the same content of the command.

2 UART Interface

2.1 General description

The serial interface between the Reader and the host controller is a full duplex interface using the two lines RX and TX.
RX is used to receive data from the host controller;
TX is used to send data to the host controller.
No flow control or supplementary line is used (no hand check).

The serial data format used is:

1	Start bit
8	Data bit
1	Stop bit, no parity
Default baudrate	115200 bps

3 Command Set

The following command bytes are available (listed in numerical order):

Command	Code
System	
SET_UR_BAUDRATE	0x01
SET_BUZZER	0x02
SET_LED	0x03
SET_HALT	0x0A
ISO 14443A (MIFARE Classic&Ultralight&NTAG)	
PICCAUTHKEY	0x16
PICCREAD_A	0x17
PICCWRITE_A	0x18
PICCWRITE_UL	0x19
PICCINITVL	0x1A
PICCVALUE_A	0x1B
PICCBAK_A	0x1C
PICCREADVL	0x1D
PICCRESET	0x21
PICCACTIVATE	0x22
PICCRATS	0x2A
PICCAPDU	0x2C
PICCTRANSFER	0x2E
MIFARE DESFire(MF3 IC D41)	
PICC_MF3_AUTHENTICATE	0x81
PICC_MF3_GETKEYSETTING	0x82
PICC_MF3_CHANGEKEY	0x83
PICC_MF3_CHANGEKEYSET	0x84
PICC_MF3_GETKEYVER	0x85
PICC_MF3_CREATEAPP	0x86
PICC_MF3_DELETEAPP	0x87
PICC_MF3_GETAPPIDS	0x88
PICC_MF3_SELECTAPP	0x89
PICC_MF3_FORMATPICC	0x8A
PICC_MF3_GETVERSION	0x8B
PICC_MF3_GETFILEIDS	0x8C
PICC_MF3_GETFILESET	0x8D
PICC_MF3_CHANGEFILESET	0x8E
PICC_MF3_CREATESTDDTFL	0x8F
PICC_MF3_CREATEBKPDTFL	0x90
PICC_MF3_CREATEVALUEFL	0x91
PICC_MF3_CREATELNRRECFL	0x92
PICC_MF3_CREATECYCRECFL	0x93
PICC_MF3_DELETEFILE	0x94
PICC_MF3_REAdDATA	0x95
PICC_MF3_WRITEDATA	0x96
PICC_MF3_GETVALUE	0x97
PICC_MF3_CREDIT	0x98

PICC_MF3_DEBIT	0x99
PICC_MF3_LIMITEDCREDIT	0x9A
PICC_MF3_WRITERECORD	0x9B
PICC_MF3_READRECORD	0x9C
PICC_MF3_CLEARRECORDFILE	0x9D
PICC_MF3_COMMITTRANS	0x9E
PICC_MF3_ABORTTRANS	0x9F
ISO 14443B	
PICCACTIVATE_B	0x41
ISO 15693	
I2_INVENTORY	0xA1
I2_READ_BLOCK	0xA3
I2_WRITE_BLOCK	0xA4
I2_LOCK_BLOCK	0xA5
I2_WRITE_AFI	0xA8
I2_LOCK_AFI	0xA9
I2_WRITE_DSFID	0xAA
I2_LOCK_DSFID	0xAB
I2_GET_SYSTEM_INFO	0xAC
I2_MultipleBlockSecurityStatus	0xAD
ICCPOWERUP_ISO	0x61
ICCPOWEROFF	0x64
ICCAPDU	0x65
ICCCHECK_PRES	0x68
ICCSETBAUDRATE	0x6B
ISO 15693	
ISO18000P3M3_INVENTORY	0xB1
ISO18000P3M3_ACK	0xB2
ISO18000P3M3_REQRN	0xB3
ISO18000P3M3_READ	0xB4
ISO18000P3M3_WRITE	0xB5
ISO18000P3M3_KILL	0xB6
ISO18000P3M3_LOCK	0xB7
ISO18000P3M3_ACCESS	0xB8
ISO18000P3M3_BLOCKWRITE	0xB9
ISO18000P3M3_BLOCKERASE	0xBA
ISO18000P3M3_BLOCKPERMALOCK	0xBB
ISO18000P3M3_SETHANDLe	0xBC

4 Error Code List

4.1 List of possible error code (Reader, System command)

Status code	Description
0xF1	LRC error
0xF2	NO THIS CMD
0xF3	SET_ERROR
0xF4	PARA_ERROR
0xB1	NO_CARD
0xB2	ANTICOLL_ERROR
0xB3	SELECT_ERROR
0xB4	HALT_ERROR
0xB6	AUTH_ERROR
0xB7	READ_ERROR
0xB8	WRITE_ERROR
0xB9	VALUEOPER_ERROR
0xBA	VALUEBAK_ERROR
0xBC	RATS_ERROR
0xBE	TPCL_ERROR
0xD1	POWERUP_ERROR
0xD2	POWEROFF_ERROR
0xD3	APDU_ERROR
0xD4	PTS_ERROR
0xD5	NO_SLOT
0xD6	CHACK_ERROR

4.2 DESFire Card Error Code (Card Return)

Hex Code	Status	Description
0X00	OPERATION_OK	Successful operation
0X0C	NO_CHANGES	No changes done to backup files, CommitTransaction /AbortTransaction not necessary
0X0E	OUT_OF_EERPROM_ERROR	Insufficient NV-Memory to complete command
0X1C	ILLEGAL_COMMAND_CODE	Command code not supported
0X1E	INTEGRITY_ERROR	CRC or MAC does not match data Padding bytes not valid
0X40	NO_SUCH_KEY	Invalid key number specified
0X7E	LENGTH_ERROR	Length of command string invalid
0X9D	PERMISSION_DENIED	Current configuration / Status does not allow the requested command
0X9E	PARAMETER_ERROR	Value of the parameter(s) invalid
0XA0	APPLICATION_NOT_FOUND	Requested AID not present on PICC
0XA1	APPL_INTEGRITEY_ERROR	Unrecoverable error within application, application will be disabled *
0XAE	AUTHENTICATION_ERROR	Current authentication status does not allow the requested command
0XAF	ADDITIONAL_FRAME	Additional data frame is expected to be sent
0XBE	BOUNDARY_ERROR	Attempt to read/write data from/to beyond the files's/record's limits

		Attempt to exceed the limits of a value file
0XC1	PICC_INTEGRITY_ERROR	Unrecoverable error within PICC, PICC will be disable*
0XCA	COMMAND_ABORTED	Previous command was not fully completed Not all Frames were requested or provided by the PCD
0XCD	PICC_DISABLED_ERROR	PICC was disabled by an unrecoverable error *
0XCE	COUNT_ERROR	Number of Applications limited to 28, no additional CreateApplication possible
0XDE	DUPLICATE_ERROR	Creation of file/application failed because file/application with same number already exists
0XEE	EEPROM_ERROR	Cound not complete NV-write operation due to loss of power, internal backup/rollback mechanism activated*
0XF0	FILE_NOT_FOUND	Specified file number does not exist
0XF1	FILE_INTEGRITY_ERROR	Unrecoverable error within file, file will be disabled *

4.3 Other Error Code

Board IF Err		
0x10	TIMEOUT_RECEIVE	No input data is received within given time, time defined
0x11	LRC_ERR	Verification of input Package checksum is failed
0x12	RX_BUFFER_FULL	Interface buffer is already full
Board Function Err		
0x30	WRITE_HARDWARE_PARAM_FAIL	Writing hardware parameter in IC's EEPROM Fail
0x31	CHECKSUM_HARDWARE_PARAM_FAIL	Checksum hardware parameter failed
0x32	HARDWARE_PARAM	
Communication Protocol Err		
0x01	IO_TIMEOUT	No reply received, e.g PICC removal
0x02	INTEGRITY_ERROR	Wrong CRC or parity detected
0x03	COLLISION_ERROR	A collision occurred
0x04	BUFFER_OVERFLOW	Attempt to write beyond buffer size
0x05	FRAMING_ERROR	Invaild frame format
0x06	RROTOCOL_ERROR	Received response violates protocol
0x07	AUTH_ERROR	Authentication error
0x08	READ_WRITE_ERROR	A Read or Write error occured in RAM/ROM or Flash
0x09	TEMPERATURE_ERROR	The RC sensors signal over heating
0x0A	RF_ERROR	Error due to RF
0x0B	INTERFACE_ERROR	An Error occured in RC communication
0x0C	LENGTH_ERROR	A length error occured
0x0D	RESOURCE_ERROR	An resource error
0x0E	TX_NAK_ERROR	TX rejected sanely by the counterpart
0x0F	RX_NAK_ERROR	RX request Rejected sanely by the counterpart
0x10	EXT_RF_ERROR	Error due to External RF
0x11	NOISE_ERROR	EMVCo EMD Noise Error
0X12	ABORTED	Used when HAL Shutdown is called
0x7F	INTERNAL_ERROR	An Internal error occured
0xAD	AUTH_DELAY	Authentication Delay
0x20	UNKNOWN_CMD_TYPE	Input command category is undefined
0x21	UNKNOWN_CMD	Input command is undefined
0x22	PARAMETER_NOT_CORRECT	Parameter is incomplete or invalid

ISO14443A Err		
0xA0	A_HALT_ERR	Error if there is a response after sending Halt command
0xA1	AUTHENT_ERR	Error if Crytol bit in Control register(Reg 0x09)is not set after preforming AUTHENT command
0xA2	NOT_AUTHENT	Error from Operating MIFARE command, i.e. Increment when cryptol bit is not set
0xA3	MIFARE_ERR	NACK (0x04 or 0x05) from MIFARE card is received
FELLICA Err		
0xC0	FELLICA_RESP_CODE_ERR	Response code mismatched
ISO15693 Err		
0xD0	FLAG_ERR	Bit Error flag in ISO15693 response is set
ISO18000-3M3 Err		
0x80	FLAG_ERR	I18000P3M3_ERR_OTHER
0x81	FLAG_ERR	I18000P3M3_ERR_MEMORY_OVERRUN
0x82	FLAG_ERR	I18000P3M3_ERR_MEMORY_LOCKED
0x83	FLAG_ERR	I18000P3M3_ERR_INSUFFICIENT_POWER
0x84	FLAG_ERR	I18000P3M3_ERR_NON_SPECIFIC
RF Communication Err		
0xE0	NO_RESPONSE	No card response within given time indicating by timeout from ASIC Timer
0xE1	FRAMING_ERR	Format of receive frame errors indicating by FramingErr bit in SIC9xx's ErrorFlag register (Reg 0x0A)
0XE2	COLLISION_ERR	Bit collision is detected indicating by CollErr bit in IC's ErrorFlag register (Reg 0x0A)
0xE3	PARITY_ERR	Parity Bit Check is invalid indicating by ParityErr bit in IC's ErrorFlag register (Reg 0x0A)
0xE4	CRC_ERR	CRC Check is invalid indicating by CRCErr bit in IC's ErrorFlag register (Reg 0x0A)
0xE5	INVALID_RESP	Response is invalid or unexpected from operation protocol
0xE6	SUBC_DET_ERR	Subcarrier from card is detected indicating by SubC_Det bit in IC's Status register (Reg 0x05); but cannot recognized following standard(available only x410

4.4 Additional Error Codes when using the Commands 0x22, 0xA1, 0x41, 0x47, 0x51, 0x52, 0xB1

Status code	Description
0x80	FAILURE
0x81	COLLISION_PENDING
0x82	EXTERNAL_RFON
0x83	EXTERNAL_RFOFF
0x84	NO_TECH_DETECTED
0x85	NO_DEVICE_RESOLVED
0x86	LPCD_NO_TECH_DETECTED
0x88	MULTI_TECH_DETECTED
0x8A	MULTI_DEVICES_RESOLVED
0x8B	DEVICE_ACTIVATED
0x8C	ACTIVE_TARGET_ACTIVED
0x8D	PASSIVE_TARGET_ACTIVED
0x8E	MERGED_SEL_RES_FOUND

0x8F	ACTIVED_BY_PEER

5 COMMANDS DESCRIPTION

5.1 System Commands

5.1.1 SET_UR_BAUDRATE (0x01)

int SetUARTBaudRate(unsigned char ucRates);

----------**DLL Explanation** ----------

ucRates: Parameter only

Parameter	Baud rate (Baud)
04	9600
03	19200
02	38400
01	57600
00	115200

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send: >> 50 00 01 01 01 51 (Set to 57600 Baud)
Return: << 50 00 01 01 01 51 (Return in old Baud rate, and then the new one will be initialized)

5.1.2 SET_BUZZER (0x02)

int SetBuzzer(unsigned char ucRates,
unsigned char ucTimes);

----------**DLL Explanation** ----------

ucRates: beep keeping times will be ucRates*50 ms and silence(500-ucRates*50)ms
ucTimes: beep ucTimes times.

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send: >> 50 00 02 02 03 04 57 (beep 4 times, every beep keep sound 150ms and silence 350ms)
Return: << 50 00 00 02 52

5.1.3 SET_LED (0x03)

int SetLed(unsigned char ucRates,
unsigned char ucTimes) ;

----------**DLL Explanation** ----------

ucRates: Shine keeping times will be ucRates*50 ms and go out (500- ucRates*50)ms
ucTimes: Flicker ucTimes times.

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send: >> 50 00 02 03 03 04 56 (flicker 4 times, every time shine150ms and go out 350ms)
Return: << 50 00 00 03 53

5.1.4 GetSoftwareVS (0x04)

int GetSoftwareVS(unsigned char *relen,unsigned char *reVS);

----------**DLL Explanation** ----------

*relent: Version length
*reuid: Version return
Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send: >> 50 00 00 04 54
Return: << 50 00 04 04 **72 18 07 24** 19

5.1.5 GetReaderUID (0x05)

int _ _stdcall GetReaderUID(unsigned char *relen,
unsigned char *reuid)

----------**DLL Explanation** ----------

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send: >> 50 00 00 05 55
Return: << 50 00 0C 05 4F 7A 6A 16 68 98 0D 5A 74 12 75 77 59
Remark: 4F 7A 6A 16 68 98 0D 5A 74 12 75 77 is the reader UID

5.1.6 SET_HALT (0x0A) – Low-Power Mode

This command sets the module into standby mode to minimize the power consumption down to app. 1 mA. Any subsequent command “wakes” the module up to normal operation.

int SetHalt(void);

----------**DLL Explanation** ----------

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send: >> 50 00 00 0A 5A
Return: << 50 00 00 0A 5A (after this return, reader will go into low power mode)

5.2 ISO14443A commands

5.2.1 PICCHALT (0x14)

This command is used to make the selected card enter HALT status. In HALT status, the card will not response request sent by reader in IDLE mode; unless reset the card or remove it from antenna field then enter again.

But this CMD will response reader’s ALL request.

Note: this command is only available for ISO14443A-3 standard cards.

unsigned char PiccHalt()

----------**DLL Explanation** ----------

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send: >> 50 00 00 14 44
Return: <<50 00 00 14 44

5.2.2 PICCAUTHKEY (0x16)

int PiccAuthKey(unsigned char auth_mode,
unsigned char addr,
unsigned char *pSnr,
unsigned char *pKey)

----------**DLL Explanation** ----------

auth_mode: 0x60 --KeyA
0x61 --KeyB
addr: block number (1 byte)
S50: 0~63
S70: 0~255
PLUS CPU (2K): 0~127
PLUS CPU (4K): 0~255
pSnr: card serial number (4 bytes)
if card Serial No more than 4 bytes, only 4 MSB needed
*pKey: 6 bytes key

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send: >>50 00 0C 16 60 04 1D B7 60 57 FF FF FF FF FF FF B3
(authenticate the card(SN=1D B7 60 57) of 0x04 block(0x01 sector) using keyA(0x60)
with 6bytes key(0xFF,0xFF,0xFF,0xFF,0xFF,0xFF))
Return: <<50 00 00 16 46

5.2.3 PICCREAD_A (0x17)

int PiccRead(unsigned char ucBlock,
unsigned char *pBuf)

----------**DLL Explanation** ----------

ucBlock: Block number (1byte)
S50: 0–63
S70: 0–255
PLUS CPU (2K): 0~127
PLUS CPU (4K): 0~255
*pBuf: Block data (16 bytes)

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send: >>50 00 01 17 04 42(Read 0x04 block)

Return: <<50 00 10 17 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 57

5.2.4 PICCWRITE_A (0x18)

int PiccWrite(unsigned char ucBlock,
unsigned char *pBuf)

----------DLL Explanation ----------

ucBlock: Block number (1 byte)
S50: 0–63
S70: 0–255
PLUS CPU (2K): 0~127
PLUS CPU (4K): 0~255
*pBuf: Data to write (16 bytes)

After one block have been authenticated successfully, the other block in the same sector need no authentication

Return: 0(OK) or Error Code

----------Protocol Example----------

Send: >>50 00 11 18 04 05 05 05 05 05 05 05 05 05 05 05 05 05 05 5D (Write 0x04 block to 16bytes 0x05)
Return: <<50 00 00 18 48

5.2.5 PICCWRITE_UL (MIFARE Ultralight) (0x19)

int PiccULWrite(unsigned char ucBlock,
unsigned char *pBuf)

----------DLL Explanation ----------

ucBlock: Block number（1 byte）
S50: 0–63
S70: 0–255
PLUS CPU (2K): 0~127
PLUS CPU (4K):） ：0～255
*pBuf: Data to write （4 bytes）

Return: 0(OK) or Error Code

----------Protocol Example----------

Send: >>50 00 05 19 04 05 05 05 05 48 (Write 0x04 block to 4bytes 0x05)
Return: <<50 00 00 19 49

5.2.6 PICC_MF0_AUTHENTICATE (MIFARE Ultralight C) (0x31)

int PiccULAuth(unsigned char *pKey);

----------DLL Explanation ----------

Input parameter:
*pKey: Master keys,
default is 16 bytes 0x00 or 49 45 4D 4B 41 45 52 42 21 4E 41 43 55 4F 59 46

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send: >>50 00 10 31 **49 45 4D 4B 41 45 52 42 21 4E 41 43 55 4F 59 46** 07
Return: <<50 00 00 31 61

5.2.7 PICC_MF0_CHANGEKEY (MIFARE Ultralight C) (0x32)

Int PiccULSetKey(unsigned char *pKey);

----------**DLL Explanation** ----------

Input parameter:
* pKey: 16bytes key to be written in.

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send: >> 50 00 10 32 **00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F** 73
Return: << 50 00 00 32 62

NOTE:
AUTHENTICATE should be at first before change key.
Other ultralight command, please refer to ISO14443A command set

5.2.8 PICCINITVL (0x1A)

int PiccInitVL(unsigned char ucBlock,
unsigned char *pBuf)

----------**DLL Explanation** ----------

ucBlock: Block number (1 byte)
S50: 0–63
S70: 0–255
PLUS CPU (2K): 0~127
PLUS CPU (4K): 0~255
*pBuf: Data to write (4bytes)
Value, Signed number, LSB first

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send: >>50 00 05 1A 05 08 00 00 00 42(Set the 0x05 block to initial value of 8)
Return: <<50 00 00 1A 4A

5.2.9 PICCVALUE_A (0x1B)

int PiccValueOper(unsigned char ucOperMode,
unsigned char ucBlock,
unsigned char *pValue,
unsigned char ucTransBlock)

----------**DLL Explanation** ----------

ucOperMode: mode (1 byte)
0xC0 –withdraw/ decrease

0xC1 –deposit/increase

ucBlock: Block number (1 byte)

S50: 0–63

S70: 0–255

PLUS CPU (2K): 0~127

PLUS CPU (4K): 0~255

*pValue: Value, Signed number, LSB first

ucTransBlock: transfer or confirm to another block (1 byte) in the same sector, the value in ucBlock will not change if ucTransBlock differ from ucBlock.

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send: >>50 00 07 1B C1 05 01 00 00 00 06 8F

(operate 0x05 block and save change to 0x06 block,0x05 block will not change)

Return: <<50 00 00 1B 4B

5.2.10 PICCBAK_A (0x1C)

Int PiccBackup(unsigned char ucBlock,
unsigned char ucTransBlock)

----------**DLL Explanation** ----------

ucBlock: Block number (1 byte)

S50: 0–63

S70: 0–255

PLUS CPU (2K): 0~127

PLUS CPU (4K): 0~255

ucTransBlock: Transfer to another block

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send: >>50 00 02 1C 06 05 4D (extract 0x06 block and backup to 0x05 block)

Return: <<50 00 00 1C 4C

5.2.11 PICCREADVL (0x1D)

int PiccReadValue(unsigned char ucBlock,
unsigned char *pBuf)

----------**DLL Explanation** ----------

ucBlock: Block number (1 byte)

S50: 0–63

S70: 0–255

PLUS CPU (2K): 0~127

PLUS CPU (4K): 0~255

*pBuf: Block value, 4Bytes, LSB first

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send: >>50 00 01 1D 05 49
Return: <<50 00 04 1D 08 00 00 00 41

5.2.12 PICCRESET (0x21)

Int PiccReset(unsigned char _ms)

----------**DLL Explanation** ----------

Input parameter:
_1ms: reset the antenna （1byte）in X ms

This means the antenna will be closed in X ms, then, and then reopened
0 is to keep antenna closed

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send: >>50 00 01 21 05 75
Return: <<50 00 00 21 71

NOTE: this command is used to save power or deactivate the card in field.

5.2.13 PICCACTIVATE (0x22)

int PiccActivate(unsigned char ucRst_1ms,
unsigned char ucReqCode,
unsigned char *pATQ,
unsigned char *pSAK,
unsigned char *pUIDLen,
unsigned char *pUID)

----------**DLL Explanation** ----------

Input parameter:
ucRst_1ms:
Refer to PiccReset command, if set, the antenna will close for ucRst_1ms(ms) first and then Open

ucReqCode:0x26 IDLE,0x52 ALL

Output variables：
*pATQ:Answer to request (ATQ) (2bytes)
*pSAK:Select acknowledge (SAK) (1byte)
*pUIDLen:UID length (1byte）
*pUID:UID (4 bytes or 7bytes most)

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send >>50 00 02 22 10 52 32
Return <<50 00 08 22 04 00 08 04 1D B7 60 57 EF
(ATQ:0400; SAK:0x08; 0x04 bytes UID: 1D B7 60 57)

NOTE: use this command to activate the card before any other command, PICCACTIVATE will run REQA, Anti-collision and Select sequence as defined in ISO/IEC 14443_3 document.

5.2.14 AUTOLISTCARD (0x23)

```
int PiccAutoListCard(    unsigned char ucType,
                         unsigned char ucPerod,
                         unsigned char ucANT,
                         unsigned char ucNotice,
                         unsigned char ucRFU)
```

----------DLL Explanation ----------

Input parameter:

TYPE:

Card type: (8 bit)

Bit0	Bit1	Bit2	Bit3	Bit4	Bit5	Bit6	Bit7
ISO14443A	ISO14443B	ISO15693	SONY Felica	Chinese ID			

TYPE is 0x01: ISO14443 A card only

TYPE is 0x04: ISO15693 Only

TYPE is 0x05: ISO15693 + ISO14443 A

TYPE is 0xFF: All card type the module supported

（Exceptional case, TYPE set to 0x00 is the same as 0xFF)

NOTE: This module support ISO15693 and ISO14443A card autolist funtion

PERIOD：

The time interval between the antenna scanning. Generally set to 100ms, that is 0x64

NOTE: If PERIOD set to 0x00, the autolistcard function will be stopping constant.

ANT：

Which ant use to list the card (8 bit)_

0x01 will use ANT1, if place the card toANT2, the module will not reply.

0x03 will use ANT1 and ANT2, the module will list the card in ANT1 field first and the change to ANT2, any ANT detected will output the card information according to he output setting.

0xFF will list card form ANT1 to ANT8

0x00 will list card cooperatively of all antenna in one time. (Default setting)

NOTE: This module support ANT1 and ANT2 only

NOTICE:

Event notification, supports 4 types of card event notification:

0x01 = NOTICE when a tag enters the field

0x02 = NOTICE when a tag leaves the field

0x03 = NOTICE when a tag enters and leaves the field

0x04 = NOTICE continuously as long as the tag is in field, notification PERIOD is defined with parameter PERIOD

RFU：

Reserved for future use.

Output variables：
NULL

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send >>50 00 05 23 FF 64 01 01 00 ED
Return <<50 00 00 23 73 (ACK of setting, but not the card reporting message)

NOTE:
1.After this command, if card in ,module TX pin will output information, example：

<<50 00 0D 23 04 64 03 01 00 47 5B 0A 3A 00 01 04 E0 D5
(47 5B 0A 3A 00 01 04 E0 is ISO 15693 card UID)

2.TYPE and PERIOD should be set, other parameter is optional

Default:
ANT: 0xFF
NOTICE: 0x01
RFU: 0x00
Example：>>50 00 02 23 00 64 15 is equal to >>50 00 05 23 00 64 FF 01 00 EC

3.other command will not stop autolistcard function ,except reset all parameter to 0x00 again or you will use following command to stop this autolistcard function temporary:

>>50 00 01 23 03 71 (Stop 3s) if the LEN0/LEN1 is 00 01, the autolist function will temporary stop in “TYPE” (the parameter after the CMD code 0x23) seconds.

Automatic reporting message format：

format：

SOF	LEN0	LEN1	CMD	TYPE	PERIOD	ANT	NOTICE	RFU	INFOR	EOF
50	00	XX	23	00-FF	00-FF	00-FF	01-04	00	XX...	XOR

Only one “INFOR” more than the set command this is the card information

ISO14443A INFOR：

ATQL	ATQH	SAK	UID Length	UID
Low byte of ATQ	High byte of ATQ	SAK	4 or 7	4 or 7 bytes UID

Example 1: Mifare 1 S50 INFOR: 04 00 08 04 11 22 33 44 （11 22 33 44 is UID）,according to he setting of CMD 50 00 05 23 05 64 03 01 00 15, if this card place to ANT1 field , the module will output >>50 00 0D 23 01 64 01 01 00 04 00 08 04 11 22 33 44 57

The 5 byte of the red font corresponds to the 5 byte of the automatic list card command, but the information is more specific:
01：TYPE, inform that this is a ISO14443A card, if 04 that is ISO15693
64：PERIOD, 100ms
01：ANT, 01 inform that the first antenna ANT1 detected this card
01：NOTICE, 01 inform that this is an Entry event
00：RFU

ISO15693 INFOR：

UID1	UID2	UID3	UID4	UID5	UID6	UID7	UID8
Xx	Xx	Xx	Xx	Xx	Xx	Xx	E0

Example 2: 15693 INFOR is 11 22 33 44 00 01 04 E0 （8bytes UID, end with 0xE0 In general）according to he setting of CMD 50 00 05 23 05 64 03 01 00 15, if this card place to ANT2 field, the module will output 50 00 0D 23 04 64 02 01 00 11 22 33 44 00 01 04 E0 BC

The 5 byte of the red font corresponds to the 5 byte of the automatic list card command, but the information is more specific:

04：TYPE, inform that this is an ISO15693 card, if 01 that is ISO14443A

64：PERIOD, 100ms

02：ANT, 02 inform that the second antenna ANT2 detected this card

01：NOTICE, 01 inform that this is an Entry event

00：RFU

5.2.15 Read Multiple Blocks from Multiple Sectors (0x27)

This command is not supported in the DLL.

Input parameter

Antenna Shutoff Time: 0x00…0xFF ms

Request All/Idle: 0x52 = use Request ALL, 0x26 = Request IDLE

Start Sector, Block number is calculated in firmware

Number of consecutive sectors.

Number of Blocks from each sector.

Output variables

Several memory blocks of 16 Bytes each.

----------Protocol Example----------

Send >> 50 00 0C 27 05 52 01 03 02 60 FF FF FF FF FF FF 4C

Return << 50 00 60 27 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD 17

The Bytes in Detail, Command from PC/PLC to RFID Device

50	START
00 0C	12 Bytes Payload
27	Command Code
05	Reset Time, Antenna Shutoff Time in ms
52	use Request ALL, 26 = Request IDLE
01	Start Sector, Block number is calculated in firmware
03	Number of consecutive Sectors
02	Number of Blocks from each Sector
60	0x60 = authenticate with Key A, 0x61 = authenticate with Key B
FF FF FF FF FF FF	Key
4C	Checksum

The Bytes in Detail, Reply from RFID Device to PC/PLC

50 Start of telegram
00 60 Number of Bytes in Payload, 0x60 =96 Bytes = 8 Blocks
27 Command Code
44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 = data from block #04 in sector #01
55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 = data from block #05 in sector #01
88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 = data from block #08 in sector #02
99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 = data from block #09 in sector #02
CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC = data from block #12 in sector #03
DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD = data from block #13 in sector #03
17 BCC

5.2.16 PICCRATS (0x2A)

```
int PiccRequestATS(     unsigned char ucRFU,
                        unsigned char *pATSLen,
                        unsigned char *pATS  )
```

----------DLL Explanation ----------

Input parameter:
ucRFU: Set to 0x00

Output variables：
* pATSLen:Length of the ATS from the card
*pATS:ATS （answer to select ）

Return: 0(OK) or Error Code

----------Protocol Example----------

Send >>50 00 00 2A 7A
Return <<50 00 10 2A 10 78 80 90 02 20 90 00 00 00 00 00 F6 D0 B1 26 11

NOTE:
Set ISO14443-3 card into ISO14443-4 mode

5.2.17 PICCAPDU (0x2C)

```
int PiccAPDU(           unsigned int usSendLen,
                        unsigned char *pSendBuf,
                        unsigned int *pRcvLen,
                        unsigned char *pRcvBuf)
```

----------DLL Explanation ----------

Input parameter:
usSendLen:length of data to be sent to the card
*pSendBuf:Data to be sent to the card

Output variables：
pRcvLen:length of data received from the card
* pRcvBuf:Data received from the card

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send >>50 00 05 2C 00 84 00 00 08 F5
Return <<50 00 0A 2C F2 EB 10 97 2D A5 54 3B 90 00 9F

NOTE:
For ISO14443A Card:
The Card have been RATS and go into ISO14443-4 mode may use this 0x2c command
For ISO14443B Card:
RATS is not need for Type B Card, so after 0x41 command, you may use this 0x2c command.

5.2.18 PICCTRANSFER (0x2E)

```
int PiccTransfer(       unsigned int usSendLen,
                        unsigned char *pSendBuf,
                        unsigned int *pRcvLen,
                        unsigned char *pRcvBuf)
```

----------**DLL Explanation** ----------

Input parameter:
usSendLen:length of data to be sent to the card
*pSendBuf:Data to be sent to the card

Output variables:
pRcvLen:length of data received from the card
* pRcvBuf:Data received from the card

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send >>50 00 07 2E 0A 00 00 84 00 00 08 FF
Return <<50 00 0C 2E 0A 00 F2 EB 10 97 2D A5 54 3B 90 00 91

NOTE:
For ISO14443A Card:
The Card is activated into ISO14443-3 (have not RATS) mode may use this 0x2E command

For ISO15693 Card:
The first CMD before transfer must be ISO15693_Inventory, this will set the reader (or module) go into ISO15693 mode
Example:
Refer to the card datasheet, you will find a command Example (a ST LRI2K card)

20 Commands codes

The LRI2K supports the commands described in this section. Their codes are given in *Table 20*.

Table 20. Command codes

Command code standard	Function
01h	Inventory
02h	Stay Quiet
20h	Read Single Block
21h	Write Single Block
22h	Lock Block
23h	Read Multiple Block
25h	Select
26h	Reset to Ready
27h	Write AFI
28h	Lock AFI
29h	Write DSFID
2Ah	Lock DSFID
2Bh	Get System Info
2Ch	Get Multiple Block Security Status

Command code custom	Function
A6h	Kill
B1h	Write Kill
B2h	Lock Kill
C0h	Fast Read Single Block
C1h	Fast Inventory Initiated
C2h	Fast Initiate
C3h	Fast Read Multiple Block
D1h	Inventory Initiated
D2h	Initiate

20.6 Read Multiple Block

When receiving the Read Multiple Block command, the LRI2K reads the selected blocks and sends back their value in multiples of 32 bits in the response. The blocks are numbered from '00 to '3F' in the request and the value is minus one (–1) in the field. For example, if the "number of blocks" field contains the value 06h, 7 blocks will be read. The maximum number of blocks is fixed at 64. During Sequential Block Read, when the block address reaches 64, it rolls over to 0. The Option_flag is supported.

Table 34. Read Multiple Block request format

Request SOF	Request_ flags	Read Multiple Block	UID	First block number	Number of blocks	CRC16	Request EOF
	8 bits	23h	64 bits	8 bits	8 bits	16 bits	

Request parameters:

- Option_flag
- UID (Optional)
- First block number
- Number of blocks

Table 35. Read Multiple Block response format when Error_flag is NOT set

Response SOF	Response_ flags	Block Locking Status	Data	CRC16	Response EOF
	8 bits	8 bits(1)	32 bits(1)	16 bits	

The Read Multiple Block request command is 02 23 00 04 (read 5 block fr 00 block)
Request flat = 0x02 (with no UID, fast mode)
Read multi = 0x23(ISO15693 card Command)
UID = (not use)

First block = 0x00
Number block = 0x04 (will read 0x05 block back)
(SOF/EOF/CRC16 is not needed, the module will handle it)
So, the Reader/Module RS232 command is 50 00 04 2E 02 23 00 04 5F

5.3 DESFire commands

MF3 IC D40 Command Set – Security Related Commands:
The MF3 IC D40 provides the following command set for security related functions:

5.3.1 PICC_MF3_AUTHENTICATE (0x81)

In this procedure both, the PICC as well as the reader device, show in an encrypted way that they possess the same secret which especially means the same key. This procedure not only confirms that both entities can trust each other but also generates a session key which can be used to keep the further communication path secure. As the name “session key” implicitly indicates, each time a new authentication procedure is successfully completed a new key for further cryptographic operations is obtained.

int DESAuthenticate(unsigned char KeyNO,
unsigned char *pKey)

----------**DLL Explanation** ----------

Input parameter:

KeyNO: Master keys is 0x00,This value in PICC level (selected AID = 0x00)and on Application level
*pKey: 16 bytes key

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send >>50 00 11 81 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 C0
Return <<50 00 00 81 D1

NOTE:
Depending on the configuration of the application (represented by its AID), an authentication has to be done to perform specific operations:
Gather information about the application
Change the keys of the application
Create and delete files within the application
Change access rights
Access data files in the authenticated application

Depending on the security configuration of the PICC, the following commands may require an authentication with the PICC master keys:
Gather information about the applications on the PICC
Change the PICC master key itself
Change the PICC key settings
Create a new application
Delete an existing application

The authentication state is invalidated by
Selecting an application

Changing the key which was used for reaching the currently valid authentication status
A failed authentication

Please note: Master keys are identified by their key number 0x00. This is valid on PICC level (selected AID = 0x00) and on Application level (selected AID ≠ 0x00).

5.3.2 PICC_MF3_GETKEYSETTING (0x82)

The GetKeySettings command allows to get configuration information on the PICC and application master key configuration settings ,In addition it returns the maximum number of keys which can be stored within the selected application.

```
int DESGetKeySetting(     unsigned char *pKeySetting,
                          unsigned char *pMaxKeyNum)
```

----------DLL Explanation ----------

Output variables:

*pKeySetting: Key settings
*pMaxKeyNum: Max No of keys

Return: 0(OK) or Error Code

----------Protocol Example----------

Send >>50 00 00 82 D2
Return <<50 00 02 82 0F 01 DE

NOTE:
Depending on the master key settings, a preceding authentication with the master key is required.
If the PICC master key settings are queried (currently selected AID = 0x00), the number of keys is returned as 0x01, as only one PICC master key exists on a PICC.

5.3.3 PICC_MF3_CHANGEKEY (0x83)

This command allows to change any key stored on the PICC
If AID = 0x00 is selected, the change applies to the PICC master key and therefore only KeyNo = 0x00 is valid (only one PICC master key is present on a PICC). In all other cases (AID ≠ 0x00) the change applies to the specified KeyNo within the currently selected application (represented by it's AID).

```
int DESChangKey(          unsigned char KeyNo,
                          unsigned char KeySettings,
                          unsigned char *pNewKey,
                          unsigned char *pOldKey)
```

----------DLL Explanation ----------

Input parameter:
KeyNo: As a parameter this command takes the KeyNo which is of one byte length and has to be in the range from 0x00 to number of application keys - 1.
KeySettings:The respective key settings (see chapter 4.3.2) define whether a change of keys is permitted or not, additionally they show which key is needed for Authentication before the ChangeKey Command.

Output variables：
*pNewKey:16bytes new key

*pOldKey:16bytes old key

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send >>50 00 22 83 01 09 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 F9
Return <<50 00 00 83 D8

NOTE: To change any key (except Master Key and the ChangeKey Key), authentication with the ChangeKey is necessary.
To change the ChangeKey Key or the Master Key, authentication with the Master Key is necessary.
After a successful change of the key used to reach the current authentication status, this authentication is invalidated i.e. an authentication with the new key is necessary for subsequent operations.

5.3.4 PICC_MF3_CHANGEKEYSET (0x84)

This command changes the master key configuration settings depending on the currently selected AID
If AID = 0x00 has been selected in advance, the change applies to the PICC key settings, otherwise (AID ≠ 0x00) it applies to the application key settings of the currently selected application.

PICC Master Key Settings:

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
RFU	RFU	RFU	RFU	configuration changeable	PICC master key not required for create / delete	free directory list access without PICC master key	allow changing the PICC master key

On PICC Level (selected AID = 0x00) the coding is interpreted as:

Bit7-Bit 4: RFU, has to be set to 0.

Bit3: codes whether a change of the PICC master key settings is allowed:

- 0: configuration not changeable anymore (frozen).
- 1: this configuration is changeable if authenticated with PICC master key (default setting).

Bit2: codes whether PICC master key authentication is needed before Create- / DeleteApplication

- 0: CreateApplication / DeleteApplication is permitted only with PICC master key authentication.
- 1: - CreateApplication is permitted without PICC master key authentication (default setting).

  - DeleteApplication requires an authentication with PICC master key or application master key*.

Bit1: codes whether PICC master key authentication is needed for application directory access:

- 0: Successful PICC master key authentication is required for executing the GetApplicationIDs and GetKeySettings commands.
- 1: GetApplicationIDs and GetKeySettings commands succeed independently of a preceding PICC master key authentication (default setting).

Bit0: codes whether the PICC master key is changeable:

- 0: PICC Master key is not changeable anymore (frozen).
- 1: PICC Master key is changeable (authentication with the current PICC master key necessary, default setting).

Application Master Key Settings:

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ChangeKey Access Rights Bit3	ChangeKey Access Rights Bit2	ChangeKey Access Rights Bit1	ChangeKey Access Rights Bit0	configuration changeable	free create / delete without master key	free directory list access without master key	allow change master key

On Application Level (selected AID ≠ 0x00) the coding is interpreted as:

Bit7-Bit4: hold the Access Rights for changing application keys (ChangeKey command).

- 0x0: Application master key authentication is necessary to change any key (default).
- 0x1 .. 0xD: Authentication with the specified key is necessary to change any key.
- 0xE: Authentication with the key to be changed (same KeyNo) is necessary to change a key.
- 0xF: All Keys (except application master key, see Bit0) within this application are frozen.

Bit3: codes whether a change of the application master key settings is allowed:

- 0: configuration not changeable anymore (frozen).
- 1: this configuration is changeable if authenticated with the application master key (default setting).

Bit2: codes whether application master key authentication is needed before CreateFile / DeleteFile

- 0: CreateFile / DeleteFile is permitted only with application master key authentication.
- 1: CreateFile / DeleteFile is permitted also without application master key authentication (default setting).

Bit1: codes whether application master key authentication is needed for file directory access:

- 0: Successful application master key authentication is required for executing the GetFileIDs GetFileSettings and GetKeySettings commands.
- 1: GetFileIDs, GetFileSettings and GetKeySettings commands succeed independently of a preceding application master key authentication (default setting).

Bit0: codes whether the application master key is changeable:

- 0: Application master key is not changeable anymore (frozen).
- 1: Application master key is changeable (authentication with the current application master key necessary, default setting).

int DESChangKeySetting(unsigned char KeySetting)

----------DLL Explanation ----------

Input parameter:
KeySettings: the new master key settings.

Return: 0(OK) or Error Code

----------Protocol Example----------

Send >>50 00 01 84 09 DC
Return <<50 00 00 84 D4

NOTE: This command only succeeds if the “configuration changeable” bit, see below, of the current key settings was not cleared before.
Additionally a successful preceding authentication with the master key is required (PICC master key ifAID = 0x00, else with application master key).
In case of usage of the application master key for deletion, the application which is about to be deleted needs to be Selected and Authenticated with the application master key prior to the DeleteApplication command.

5.3.5 PICC_MF3_GETKEYVER (0x85)

The GetKeyVersion command allows to read out the current key version of any key stored on the PICC.
If AID = 0x00 is selected, the command returns the version of the PICC master key and therefore only KeyNo = 0x00 is valid (only one PICC master key is present on a PICC). In all other cases (AID ≠ 0x00) the version of the specified KeyNo within the currently selected application (represented by it's AID) is returned.

int DESGetKeyVersion(unsigned char KeyNO, unsigned char *pKeyVersion)

----------**DLL Explanation** ----------

Input parameter:
KeyNO: key number

Output variables:
*pKeyVersion: returns the current version of the specified key as an unsigned byte.

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send >>50 00 01 85 00 D4
Return <<50 00 01 85 00 64

NOTE: This command can be issued without valid authentication.

5.3.6 PICC_MF3_CREATEAPP (0x86)

The CreateApplication command allows to create new applications on the PICC.

int DESCreateApplication(unsigned long AID,
unsigned char KeySetting,
unsigned char KeyNo)

----------**DLL Explanation** ----------

Input parameter:
AID: 3bytes LSB, a 24 bit number. Application Identifier
0x00 00 00 is reserved as a reference to the PICC itself.
KeySetting: the Application Master Key Settings as defined in PICC_MF3_CHANGEKEYSET
KeyNo: defines how many keys can be stored within the application for cryptographic purposes.

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send >>50 00 05 86 01 00 00 09 04 DF
Return <<50 00 00 86 D6

NOTE: Depending on the PICC master key settings, a preceding PICC master key authentication may be required.

This command requires that the currently selected AID is 0x00 00 00 which references the card level.
One PICC can hold up to 28 Applications. Each application is linked to a set of up to 14 different user definable access keys. To store data in an application, it is necessary to create so called files within that application. Up to 16 files of different size

and type can be created within each application. Different levels of access rights for each single file can be linked to the keys of the application.

All keys are initialized with a string consisting of sixteen 0x00 bytes and therefore are single DES keys by definition

It is strongly recommended to personalize the keys latest at card issuing using the command ChangeKey.

5.3.7 PICC_MF3_DELETEAPP (0x87)

The DeleteApplication command allows to permanently deactivate applications on the PICC

int DESDeleteApplication(unsigned long AID)

----------**DLL Explanation** ----------

Input parameter:
AID: 3bytes LSB, a 24 bit number. Application Identifier 0x00 00 00 is reserved as a reference to the PICC itself.

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send >>50 00 03 87 01 00 00 D5
Return <<50 00 00 87 D7

NOTE:
Depending on the PICC master key settings, either a preceding PICC master key authentication or an application master key authentication is required.

In the latter case, it has to be the master key authentication for the application which shall be deleted by this command.

Even if the PICC master key contains the default value 0 and the bit “free create / delete without PICC master key” is set, it is necessary to be either authenticated with the zero PICC master key or the respective application master key.

If the currently selected application is deleted, this command automatically selects the PICC level, selected AID = 0x00 00 00.

5.3.8 PICC_MF3_GETAPPIDS (0x88)

The GetApplicationIDs command returns the Application IDentifiers of all active applications on a PICC.

int DESGetApplicationIDs(unsigned char *pAIDs, unsigned char *pAIDno)

----------**DLL Explanation** ----------

Output variables：
* pAIDs: 3bytes LSB, As response the PICC sends a sequence of all installed AIDs.
*pAIDno: number of AID

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send >>50 00 00 88 D8
Return <<50 00 01 88 00 D9 //No app (example 1)

<<50 00 04 88 01 06 00 00 DB // One app,AID=0x000006 LSB (example 2)
<<50 00 0D 88 04 01 00 00 02 00 00 03 00 00 01 00 03 D3 //4 app (example 3)

NOTE:
This command requires that the currently selected AID is 0x00 00 00 which references the card level.
Depending on the PICC master key settings a successful authentication with the PICC master key might be required to execute this command.

5.3.9 PICC_MF3_SELECTAPP (0x89)

The SelectApplication command allows to select one specific application for further access.

int DESSelectApplication(long AID)

----------DLL Explanation ----------

Input parameter:
AID: 3bytes LSB, a 24 bit number. Application Identifier 0x00 00 00 is reserved as a reference to the PICC itself.

Return: 0(OK) or Error Code

----------Protocol Example----------

(example 1)
Send >>50 00 03 89 00 00 00 DA (AID is 0x000000)
Return <<50 00 00 89 D9

(example 2)
Send >>50 00 03 89 01 00 00 DB (AID is 0x000001)
Return <<50 00 00 89 D9

NOTE:
If this parameter is 0x00 00 00, the PICC level is selected and any further operations (typically commands like CreateApplication, DeleteApplication…) are related to this level.

If an application with the specified AID is found in the application directory of the PICC, the subsequent commands interact with this application.

As mentioned in the description of the Authenticate command, each SelectApplication command invalidates the current authentication status.

5.3.10 PICC_MF3_FORMATPICC (0x8A)

This command releases the PICC user memory.

int DESFormatPicc(void)

----------DLL Explanation ----------

Return: 0(OK) or Error Code

----------Protocol Example----------

Send >>50 00 00 8A DA
Return <<50 00 00 8A DA

NOTE:
The FormatPICC Command releases all allocated user memory on the PICC.

All applications are deleted and all files within those applications are deleted.

The PICC master key and the PICC master key settings keep their currently set values, they are not influenced by this command.

This command always requires a preceding authentication with the PICC master key.

5.3.11 PICC_MF3_GETVERSION (0x8B)

The GetVersion command returns manufacturing related data of the PICC.

int DESGetDESVersion(unsigned char *relen,unsigned char *rebuf)

----------DLL Explanation ----------

Output variables:
*relen: The length of frames
*rebuf: Three frames of manufacturing related data are returned by the PICC

Return: 0(OK) or Error Code

----------Protocol Example----------

Send >>50 00 00 8B DB
Return <<50 00 1C 8B 04 01 01 01 00 16 05 04 01 01 01 04 16 05 04 28 69 9A 4F 22 80 BA 24 17 A9 20 07 11 E7
(D21, Example 1)
<<50 00 1C 8B 04 01 01 01 00 1A 05 04 01 01 01 03 1A 05 04 41 7B D1 46 1C 80 CF B6 D4 66 90 53 08 F1
(D81, Example 2)
<<50 00 1C 8B 04 01 01 01 00 18 05 04 01 01 01 03 18 05 04 84 4D 42 78 1F 80 BA 95 D9 4D 10 40 09 4E
(D41, Example 3)

NOTE:
Three frames of manufacturing related data are returned by the PICC:
Frame1: contains hardware related information:
byte1: codes the vendor ID (0x04 for PHILIPS)
byte2: codes the type (here 0x01)
byte3: codes the subtype (here 0x01)
byte4: codes the major version number
byte5: codes the minor version number
byte6: codes the storage size* (here 0x18 = 4096 bytes)
byte7: codes the communication protocol type (here 0x05 meaning ISO 14443-2 and -3)

Frame2 contains software related information:
byte1: codes the vendor ID (here 0x04 for PHILIPS)
byte2: codes the type (here 0x01)
byte3: codes the subtype (here 0x01)
byte4: codes the major version
byte5: codes the minor version
byte6: codes the storage size* (here 0x18 = 4096 bytes)

byte7: codes the communication protocol type (here 0x05 meaning ISO 14443-3 and -4)

Frame3 returns the unique serial number, batch number, year and calendar week of production:
byte1 to byte7: code the unique serial number
byte8 to byte12: code the production batch number
byte13: codes the calendar week of production
byte14: codes the year of production

* The 7 MSBits (= n) code the storage size itself based on 2^n, the LSBit is set to '0' if the size is exactly 2^n and set to '1' if the storage size is between 2^n and 2^(n+1). For this version of DESFire the 7 MSBits are set to 0x0C (2^12 = 4096) and the LSBit is '0'.

MF3 IC D40 Command Set – Application Level Commands:
The MF3 IC D40 provides the following command set for Application level functions.

5.3.12 PICC_MF3_GETFILEIDS (0x8C)

The GetFileIDs command returns the File IDentifiers of all active files within the currently selected application.

int DESGetFileIDs(unsigned char *pIDs, unsigned char *pFileIDs)

----------DLL Explanation ----------

Output variables:
* pIDs: No of ID
* pFileIDs: Each File ID is coded in one byte and is in the range from 0x00 to 0x0F.

Return: 0(OK) or Error Code

----------Protocol Example----------

Send >>50 00 00 8C DC
Return <<50 00 01 8C 00 DD (No file, Example 1)
<<50 00 02 8C 01 05 DA (1 file - 0x05, Example 2)
<<50 00 03 8C 02 05 06 DE (two files, 05 and 06, Example 1)

NOTE:
Depending on the application master key settings (see chapter 4.3.2), a preceding authentication with the application master key might be required.

As the number of files is limited to sixteen within one application, the response always fits into one single data frame.

5.3.13 PICC_MF3_GETFILESET (0x8D)

The GetFileSettings command allows to get information on the properties of a specific file. The information provided by this command depends on the type of the file which is queried.

int DESGetFileSettings(unsigned char FileID, unsigned char *pInfoLen, unsigned char *pFileInfo)

----------DLL Explanation ----------

Input parameter:
FileID: the file number of the file to be queried within the currently selected application. This file number must be in the range between 0x00 and 0x0F.

Output variables：

* pInfoLen: File information length

* pFileInfo: File information, The first byte indicates the file's type, The next byte provides information on the file's communication settings (plain/MACed/Enciphered), This information is followed by the 2 byte file Access Rights field.

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send >>50 00 01 8D 05 D9

Return <<50 00 08 8D 07 00 00 EE EE 00 01 00 D3 //LSB File Size is 3bytes,the last byes did not needed

NOTE:

All subsequent bytes in the response have a special meaning depending on the file type:

Standard Data Files and Backup Data Files:

One field of three bytes length returns the user file size in bytes.

Value Files:

Three fields, each of four bytes length, are returned whereby the first field returns the "lower limit" of the file (as defined at file creation), the second field returns the "upper limit" (as defined at file creation) and the next field returns the current maximum "limited credit" value. If the limited credit functionality is not in use, the last field contains all zeros. The last byte codes, if the LimitedCredit command is allowed for this file (0x00 for disabled, 0x01 for enabled).

Linear Record Files and Cyclic Record Files:

Three fields, each of three bytes length, are returned whereby the first field codes the size of one single record (as defined at file creation), the second field codes the maximum number of records within the record file (as defined at file creation) and the last field returns the current number of records within the record file. This number equals the maximum number of records which currently can be read.

Coding of File Types

The files within an application can be of different types as:

- Standard Data Files (coded as 0x00)
- Backup Data Files (coded as 0x01)
- Value Files with Backup (coded as 0x02)
- Linear Record Files with Backup (coded as 0x03)
- Cyclic Record Files with Backup (coded as 0x04)

Coding of Communication Settings – Encryption Modes

The Communication Settings define the level of security for the communication between PCD and PICC. Communication Settings always apply on file-level.

The Settings are coded into one byte which needs to be set to

Communication Mode	bit 7 – bit 2	bit 1	bit 0
Plain communication	RFU = 0	ignored	0
Plain communication secured by DES/3DES MACing	RFU = 0	0	1
Fully DES/3DES enciphered communication	RFU = 0	1	1

Both DES and 3DES keys are stored in strings consisting of 16 bytes:

If the 2nd half of the key string is equal to the 1st half, the key is handled as a single DES key by the PICC.

If the 2nd half of the key string is not equal to the 1st half, the key is handled as a 3DES key.

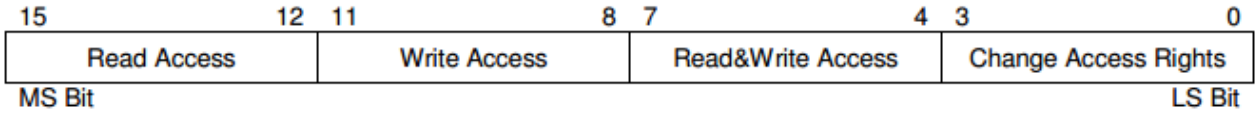
* All bytes 0x00 is the default key of the MF3 IC D40 IC, and defines single DES operation as default

All operations based on keys are executed with the respective method DES or 3DES.

Coding of Access Rights:
There are four different Access Rights (2 bytes for each file) stored for each file within each application:

- Read Access (GetValue, Debit for Value files)
- Write Access (GetValue, Debit, LimitedCredit for Value files)
- Read&Write Access (GetValue, Debit, LimitedCredit, Credit for Value files)
- ChangeAccessRights

Each of the Access Rights is coded in 4 bits, one nibble. Each nibble represents a link to one of the keys stored within the respective application's key file.
One nibble allows to code 16 different values. If such a value is set to a number between 0 and 13 (max. 14 keys), this references a certain key within the application's key file, provided that the key exists (selecting a non-existing key is not allowed).
If the number is coded as 14 (0xE) this means “free” access. Thus the regarding access is granted always with and without a preceding authentication, directly after the selection of the respective application.
The number 15 (0xF) defines the opposite of “free” access and has the meaning “deny” access. Thereforethe respective linked Access Rights is always denied.
The most significant 4 bits of the two bytes parameter code the reference number of the key which is necessary to know for getting Read Access (in case of Value files: for the GetValue and the Debit Command).
The next 4 bits hold the number of the key for getting Write Access (in case of Value files: GetValue, Debit and LimitedCredit Command).
The upper nibble of the lower byte holds the key number for getting Read&Write Access, in Value files this right allows full access (in case of Value files: GetValue, Debit, LimitedCredit and Credit Command; in case of Record files additionally: ClearRecordFile).
The least significant nibble holds the reference number of the key, which is necessary to be authenticated with in order to change the Access Rights for the file and to link each Access Right to key numbers.
Access rights are always packed in 2 bytes as follows:



Read is possible with Read Access and Read&Write Access.
Write is possible with Write Access and Read&Write Access.
If a file is accessed without valid authentication but free access (0xE) is possible through at least one relevant access right, the communication mode is forced to plain.
If only one of the keys for “Read” and “Read&Write” access (or "Write" and "Read&Write" access) is set to 0xE, the other key is different from 0xE, communication is done MACed/enciphered in case of a valid authentication and done in plain in case of no valid authentication. In the second case the communication settings, see chapter 3.2, are ignored by the PICC.

5.3.14 PICC_MF3_CHANGEFILESET (0x8E)

This command changes the access parameters of an existing file.

```
int DESChangeFileSettings(
                        unsigned char FileID,
                        unsigned char NewComSet,
                        unsigned short NewAccessRights)
```

----------DLL Explanation ----------

Input parameter:
FileID: the file number of the file to be queried within the currently selected application. This file number must be in the range between 0x00 and 0x0F.

NewComSet: the new communication settings, refer to Coding of Communication Settings – Encryption Modes
NewAccessRights: a two byte field defines the new Access Rights, please refer to Coding of Access Rights:

Return: 0(OK) or Error Code

----------Protocol Example----------

Send >>50 00 04 8E 05 00 EE EE DF
Return <<50 00 00 8E DE

NOTE:
To guarantee that the ChangeFileSettings command is coming from the same party which did the preceding authentication, it is necessary to apply basically the same security mechanism as used with the ChangeKey command.

However, if the ChangeAccessRights Access Rights is set with the value “free”, no security mechanism is necessary and therefore the data is sent as plain text (5 byte overall length).

5.3.15 PICC_MF3_CREATESTDDTFL (0x8F)

The CreateStdDataFile command is used to create files for the storage of plain unformatted user data within an existing application on the PICC.

```
int DESCreateStdDataFile(
                        unsigned char FileID,
                        unsigned char ComSet,
                        unsigned short AccessRights,
                        unsigned short FileSize)
```

----------DLL Explanation ----------

Input parameter:
FileID: the file number of the file to be queried within the currently selected application. This file number must be in the range between 0x00 and 0x0F.The file will be created in the currently selected application. It is not necessary to create the files within the application in a special order. If a file with the specified number already exists within the currently selected application, an error code is returned.

ComSet: the new communication settings, refer to Coding of Communication Settings – Encryption Modes
AccessRights: LSB, two byte field defines the new Access Rights, please refer to Coding of Access Rights:
FileSize: LSB, two byte length specifies the size of the file in bytes

Return: 0(OK) or Error Code

----------Protocol Example----------

Send >>50 00 06 8F 05 00 EE EE 00 01 DD
Return <<50 00 00 8F DF

NOTE:
The MF3 IC D40 internally allocates NV-memory in multiples of 32 bytes. Therefore a file creation command with FileSize parameter 0x00 01 (1 byte file size) will internally consume the same amount of NV-memory as a 0x00 00 20 (32 byte file size), namely 32 bytes.

5.3.16 PICC_MF3_CREATEBKPDTFL (0x90)

The CreateBackupDataFile command is used to create files for the storage of plain unformatted user data within an existing application on the PICC, additionally supporting the feature of an integrated backup mechanism.

int DESCreateBackupDataFile(
unsigned char FileID,
unsigned char ComSet,
unsigned short AccessRights,
unsigned short FileSize)

----------**DLL Explanation** ----------

Input parameter:
FileID: the file number of the file to be queried within the currently selected application. This file number must be in the range between 0x00 and 0x07., only FileID 0x00 to 0x07 is allowed.
ComSet: the new communication settings, refer to Coding of Communication Settings – Encryption Modes
AccessRights: LSB, two byte field defines the new Access Rights, please refer to Coding of Access Rights:
FileSize: LSB, two byte length specifies the size of the file in bytes

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send >>50 00 06 90 05 00 EE EE 00 01 C2
Return <<50 00 00 90 C0

NOTE:
As the name “BackupDataFile” implies, files of this type feature an integrated backup mechanism.
Every Write command is done in a independent mirror image of this file. To validate a write access to this file type, it is necessary to confirm it with a CommitTransaction command. If no CommitTransaction command is send by the PCD, only the mirror image is changed, the original data remains unchanged and valid.
Due to the mirror image a BackupDataFile always consumes DOUBLE the NV-memory on the PICC compared to a StdDataFile with the same specified FileSize.

5.3.17 PICC_MF3_CREATEVALUEFL (0x91)

The CreateValueFile command is used to create files for the storage and manipulation of 32bit signed integer values within an existing application on the PICC.

int DESCreateValueFile(
unsigned char FileID,
unsigned char ComSet,
unsigned short AccessRights,
long LowerLimit,
long UpperLimit,
long Value,
unsigned char LimitCredit)

----------**DLL Explanation** ----------

Input parameter:
FileID: As first parameter one byte codes the file number in the range 0x00 to 0x07 which the new created fileshould get within the currently selected application.

ComSet: the new communication settings, refer to Coding of Communication Settings – Encryption Modes

AccessRights: LSB, two byte field defines the new Access Rights, please refer to Coding of Access Rights:

LowerLimit: LSB, 4 byte length and codes the lower limit which is valid for this file. The lower limit marks the boundary which must not be passed by a Debit calculation on the current value. The lower limit is a 4 byte signed integer and thus may be negative too.

UpperLimit: LSB, 4 bytes are used to code the upper limit which sets the boundary in the same manner but for the Credit operation, see chapter 4.6.4. This parameter is also a 4 byte signed integer.

Value: LSB, 4 byte signed integer again and specifies the initial value of the value file. The upper and lower limit is checked by the PICC, in case of inconsistency the file is not created and an error message is sent by the PICC.

LimitCredit: Here 0x00 means that LimitedCredit is disabled and 0x01 enables this feature.

Return: 0(OK) or Error Code

----------Protocol Example----------

Send >>50 00 11 91 05 00 EE EE 00 00 00 00 FF FF 00 00 00 01 00 00 01 D5
Return <<50 00 00 91 C1

NOTE:
The upper limit has to be ≥ lower limit, otherwise an error message would be sent by the PICC and thus the file would not be created.

ValueFiles feature always the integrated backup mechanism. Therefore every access changing the value needs to be validated using the CommitTransaction command.

5.3.18 PICC_MF3_CREATELNRRECFL (0x92)

The CreateLinearRecordFile command is used to create files for multiple storage of structural data, for example for loyalty programs, within an existing application on the PICC. Once the file is filled completely with data records, further writing to the file is not possible unless it is cleared, see command ClearRecordFile.

```
int DESCreateLinearRecordFile(
                    unsigned char FileID,
                    unsigned char ComSet,
                    unsigned short AccessRights,
                    unsigned short FileSize,
                    unsigned short RecordsNum)
```

----------DLL Explanation ----------

Input parameter:
FileID: As first parameter one byte codes the file number in the range 0x00 to 0x07 which the new created fileshould get within the currently selected application.

ComSet: the new communication settings, refer to Coding of Communication Settings – Encryption Modes

AccessRights: LSB, two byte field defines the new Access Rights, please refer to Coding of Access Rights:

FileSize: LSB,two bytes length and codes the size of one single record in bytes. This parameter has to be in the range from 0x00 01 to 0xFF FF.

RecordsNum: LSB, Thus the entire file size in the PICC NV-memory is given by FileSize * RecordsNum.

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send >>50 00 08 92 05 00 EE EE 00 01 04 00 C5
Return <<50 00 00 92 C2

NOTE:
Linear Record Files feature always the integrated backup mechanism. Therefore every access appending arecord needs to be validated using the CommitTransaction command.

5.3.19 PICC_MF3_CREATECYCRECFL (0x93)

The CreateCyclicRecordFile command is used to create files for multiple storage of structural data, for example for logging transactions, within an existing application on the PICC. Once the file is filled completely with data records, the PICC automatically overwrites the oldest record with the latest written one. This wrap is fully transparent for the PCD.

```
int DESCreateCyclicRecordFile(
                    unsigned char FileID,
                    unsigned char ComSet,
                    unsigned short AccessRights,
                    unsigned short FileSize,
                    unsigned short RecordsNum)
```

----------**DLL Explanation** ----------

Input parameter:
FileID: As first parameter one byte codes the file number in the range 0x00 to 0x07 which the new created fileshould get within the currently selected application.

ComSet: the new communication settings, refer to Coding of Communication Settings – Encryption Modes

AccessRights: LSB, two byte field defines the new Access Rights, please refer to Coding of Access Rights:

FileSize: LSB,two bytes length and codes the size of one single record in bytes. This parameter has to be in the range from 0x00 01 to 0xFF FF.

RecordsNum: LSB, Thus the entire file size in the PICC NV-memory is given by FileSize * RecordsNum.

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send >> 50 00 08 93 05 00 EE EE 00 01 04 00 C4
Return << 50 00 00 93 C3

NOTE:

Cyclic Record Files feature always the integrated backup mechanism. Therefore every access appending arecord needs to be validated using the CommitTransaction command.

As the backup feature consumes one record, the 'Max. Num Of Records' needs to be one larger than the application requires.

5.3.20 PICC_MF3_DELETEFILE (0x94)

The DeleteFile command permanently deactivates a file within the file directory of the currently selected application.

int DESDeleteDESFile(unsigned char FileID)

----------**DLL Explanation** ----------

Input parameter:
FileID: This command takes one byte as parameter coding the file number which is to be in the range from 0x00 to 0x0F

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send >>50 00 01 94 05 C0
Return <<50 00 00 94 C4

NOTE:
The operation of this command invalidates the file directory entry of the specified file which means that the file can't be accessed anymore.
Depending on the application master key settings, a preceding authentication with the application master key is required.
Allocated memory blocks associated with the deleted file are not set free. The FileNo of the deleted file can be re-used to create a new file within that application.
To release memory blocks for re-use, the whole PICC user NV-memory needs to be erased using the FormatPICC command.

MF3 IC D40 Command Set – Data Manipulation Commands
The MF3 IC D40 provides the following command set for Data Manipulation.

5.3.21 PICC_MF3_REAdDATA (0x95)

The ReadData command allows to read data from Standard Data Files or Backup Data Files.

```
int DESReadData (
        unsigned char FileID,
        unsigned short Offset,
        unsigned short Length,
        unsigned char *pBuf,
        unsigned short *RcvLen)
```

----------**DLL Explanation** ----------

Input parameter:
FileID: This command takes one byte as parameter coding the file number which is to be in the range from 0x00 to 0x0F
Offset:LSB, Two byte length and codes the starting position for the read operation within the file (= offset value). This parameter has to be in the range from 0x00 00 to file size -1..
Length: LSB, is also Two byte long and specifies the number of data bytes to be read. This parameter can be in the range from 0x00 00 to 0xFF FF.

Output variables：
*pBuf:Data return
*RcvLen:length of Data return

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send >>50 00 05 95 05 00 00 05 00 C0
Return <<50 00 05 95 01 02 03 04 05 DE

NOTE:
If Backup Data Files are read after writing to them, but before issuing the CommitTransaction command, the ReadData command will always retrieve the old, unchanged data stored in the PICC. All data written to a Backup Data File is validated and externally “visible” for a ReadData command only after a CommitTransaction command.

The Read command requires a preceding authentication either with the key specified for “Read” or “Read&Write” access.

5.3.22 PICC_MF3_WRITEDATA (0x96)

The WriteData command allows to write data to Standard Data Files and Backup Data Files.

```
int DESWriteData (
        unsigned char FileID,
        unsigned short Offset,
        unsigned short Length,
        unsigned char *pBuf)
```

----------**DLL Explanation** ----------

Input parameter:
FileID: 1 byte length and defines the file number to be written to; valid range is 0x00 to 0x0F for Standard Data Files and 0x00 to 0x07 for Backup Data Files, respectively.
Offset:LSB, Two byte length and codes the starting position for the read operation within the file (= offset value). This parameter has to be in the range from 0x00 00 to file size -1..
Length: LSB, is also Two byte long and specifies the number of data bytes to be read. This parameter can be in the range from 0x00 00 to 0xFF FF.
*pBuf:Data to be write

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send >>50 00 0A 96 05 00 00 05 00 01 02 03 04 05 CD
Return <<50 00 00 96 DE

NOTE:
The Write command requires a preceding authentication either with the key specified for “Write” or “Read&Write” access

If the WriteData operation is performed on a Backup Data File, it is necessary to validate the written data with a CommitTransaction command, see chapter 4.6.10. An AbortTransaction command will invalidate all changes.

If data is written to Standard Data Files (without integrated backup mechanism), data is directly programmed into the visible NV-memory of the file. The new data is immediately available to any following ReadData commands performed on that file.

Getting an Integrity Error when writing on a Standard Data File can corrupt the content of the file.

5.3.23 PICC_MF3_GETVALUE (0x97)

The GetValue command allows to read the currently stored value from Value Files.

int DESGetValue(unsigned char FileID, long *Value)

----------DLL Explanation ----------

Input parameter:
FileID: The only parameter sent with this command is of one byte length and codes the file number. This parameter has to be in the range from 0x00 to 0x07.

Output variables:
* Value: Value to be return,LSB.

Return: 0(OK) or Error Code

----------Protocol Example----------

Send >>50 00 01 97 05 C3
Return <<50 00 04 97 01 02 03 04 C7

NOTE:
The value is always represented LSB first.
The GetValue command requires a preceding authentication with the key specified for Read, Write or Read&Write access.
After updating a value file's value but before issuing the CommitTransaction command, the GetValue command will always retrieve the old, unchanged value which is still the valid one.

5.3.24 PICC_MF3_CREDIT (0x98)

The Credit command allows to increase a value stored in a Value File.

int DESOperateValue(unsigned char FileID, unsigned char ValueCommand, long Value)

----------DLL Explanation ----------

Input parameter:
FileID: The only parameter sent with this command is of one byte length and codes the file number. This parameter has to be in the range from 0x00 to 0x07.
ValueCommand: 0x0C for CREDIT
* Value: Value to be CREDIT, LSB.

Return: 0(OK) or Error Code

----------Protocol Example----------

Send >>50 00 05 98 05 01 00 00 00 C9 //LSB
Return <<50 00 00 98 C8

NOTE:

The value is always represented LSB first.

It is necessary to validate the updated value with a CommitTransaction command. AnAbortTransaction command will invalidate all changes

The value modifications of Credit, Debit and LimitedCredit commands are cumulated until a CommitTransaction command is issued.

Credit commands do NEVER modify the Limited Credit Value of a Value file. However, if the Limited Credit Value needs to be set to 0, a LimitedCredit with value 0 can be used.

The Credit command requires a preceding authentication with the key specified for “Read&Write” access.

5.3.25 PICC_MF3_DEBIT (0x99)

The Debit command allows to decrease a value stored in a Value File.

int DESOperateValue(
unsigned char FileID,
unsigned char ValueCommand,
ong Value)

----------DLL Explanation ----------

Input parameter:
FileID: The only parameter sent with this command is of one byte length and codes the file number. This parameter has to be in the range from 0x00 to 0x07.
ValueCommand: 0xDC for DEBIT
* Value: Value to be DEBIT, LSB.

Return: 0(OK) or Error Code

----------Protocol Example----------

Send >>50 00 05 99 05 01 00 00 00 C8 //LSB
Return <<50 00 00 99 C9

NOTE:
The value is always represented LSB first.
It is necessary to validate the updated value with a CommitTransaction command, AnAbortTransaction command will invalidate all changes.

The value modifications of Credit, Debit and LimitedCredit commands are cumulated until a CommitTransaction command is issued.

The Debit command requires a preceding authentication with one of the keys specified for Read, Write or Read&Write access.

If the usage of the LimitedCredit feature is enabled, the new limit for a subsequent LimitedCredit command is set to the sum of Debit commands within one transaction before issuing a CommitTransaction command. This assures that a LimitedCredit command can not re-book more values than a debiting transaction deducted before.

5.3.26 PICC_MF3_LIMITEDCREDIT (0x9A)

The LimitedCredit command allows a limited increase of a value stored in a Value File without having full Read&Write permissions to the file. This feature can be enabled or disabled during value file creation.

```
int DESOperateValue(
        unsigned char FileID,
        unsigned char ValueCommand,
        long Value)
```

----------DLL Explanation ----------

Input parameter:
FileID: The only parameter sent with this command is of one byte length and codes the file number. This parameter has to be in the range from 0x00 to 0x07.
ValueCommand: 0x1C for LIMITEDCREDIT
* Value: Value to be LIMITEDCREDIT,LSB.

Return: 0(OK) or Error Code

----------Protocol Example----------

Send >>50 00 05 9A 05 01 00 00 00 C8
Return <<50 00 00 9A C9

NOTE:
The value is always represented LSB first.
It is necessary to validate the updated value with a CommitTransaction command, AnAbortTransaction command will invalidate all changes

The value modifications of Credit, Debit and LimitedCredit commands are cumulated until a CommitTransaction command is issued.

The LimitedCredit command requires a preceding authentication with the key specified for “Write” or "Read &Write" access.

The value for LimitedCredit is limited to the sum of the Debit commands on this value file within the most recent transaction containing at least one Debit. After executing the LimitedCredit command the new limit is set to 0 regardless of the amount which has been re-booked. Therefore the LimitedCredit command can only be used once after a Debit transaction.

5.3.27 PICC_MF3_WRITERECORD (0x9B)

The WriteRecord command allows to write data to a record in a Cyclic or Linear Record File.

```
int DESWriteRecord(
        unsigned char FileID,
        unsigned short Offset,
        unsigned short Length,
        unsigned char *pBuf)
```

----------DLL Explanation ----------

Input parameter:
FileID: This parameter has to be in the range from 0x00 to 0x07.

Offset: LSB,two bytes code the offset within one single record (in bytes). This parameter has to be in the range from 0x00 00 to record size - 1.

Length: LSB, the length of data which is to be written to the record file. This parameter has to be in the range from 0x00 01 to record size - offset.

*pBuf: the data which is to be written to the record file.

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send >>50 00 09 9B 05 00 00 04 00 11 22 33 44 87
Return <<50 00 00 9B CB

NOTE:
The WriteRecord command appends one record at the end of the linear record file, it erases and overwrites the oldest record in case of a cyclic record file if it is already full. The entire new record is cleared before data is written to it.
If no CommitTransaction command is sent after a WriteRecord command, the next WriteRecord command to the same file writes to the already created record. After sending a CommitTransaction command, a new WriteRecord command will create a new record in the record file. An AbortTransaction command will invalidate all changes.
After issuing a ClearRecordFile command, but before a CommitTransaction / AbortTransaction command, a WriteRecord command to the same record file will fail.
The WriteRecord command requires a preceding authentication either with the key specified for “Write” or “Read&Write” access.

5.3.28 PICC_MF3_READRECORD (0x9C)

The ReadRecords command allows to read out a set of complete records from a Cyclic or Linear Record File.

```
int DESReadRecord(
        unsigned char FileID,
        unsigned short RecordNo,
        unsigned short RecordNum,
        unsigned char *pBuf,
        unsigned short *RcvLen)
```

----------**DLL Explanation** ----------

Input parameter:
FileID: This parameter has to be in the range from 0x00 to 0x07.
RecordNo: LSB, two bytes long and codes the offset of the newest record which is read out. In case of 0x00 00 the latest record is read out. The offset value must be in the range from 0x00 to number of existing records – 1.
RecordNum: the number of records to be read from the PICC. Records are always transmitted by the PICC in chronological order (= starting with the oldest, which is number of records – 1 before the one addressed by the given offset). If this parameter is set to 0x00 00 then all records, from the oldest record up to and including the newest record (given by the offset parameter) are read.

*pBuf: the data return in the record file.
*RcvLen: length of the data return.

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send >>50 00 05 9C 05 00 00 04 00 C8
Return <<50 00 05 9C 01 02 03 04 05 C8

NOTE:
In cyclic record files the maximum number of stored valid records is one less than the number ofrecords specified in the CreateCyclicRecordFile command.

A ReadRecords command on an empty record file (directly after creation or after a committed clearance) will result in an error.

The ReadRecords command requires a preceding authentication either with the key specified for “Read” or “Read&Write” access.

5.3.29 PICC_MF3_CLEARRECORDFILE (0x9D)

The ClearRecordFile command allows to reset a Cyclic or Linear Record File to the empty state.

int DESClearRecordFile(unsigned char FileID)

----------**DLL Explanation** ----------

Input parameter:
FileID: This parameter has to be in the range from 0x00 to 0x07.

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send >>50 00 01 9D 05 C9
Return <<50 00 00 9D CD

NOTE:
After executing the ClearRecordFile command but before CommitTransaction, all subsequent WriteRecord commands, will fail. The ReadRecords command, will return the old still valid records.
After the CommitTransaction command is issued, a ReadRecords command will fail, WriteRecord commands will be successful.
An AbortTransaction command (instead of CommitTransaction) will invalidate the clearance.
Full “Read&Write” permission on the file is necessary for executing this command.

5.3.30 PICC_MF3_COMMITTRANS (0x9E)

The CommitTransaction command allows to validate all previous write access on Backup Data Files, ValueFiles and Record Files within one application.

int DESTransaction(unsigned char Command)

----------**DLL Explanation** ----------

Input parameter:
Command: 0xC7 for CommitTransaction

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send >>50 00 00 9E CE
Return <<50 00 00 9E CE

The CommitTransaction command validates all write access to files with integrated backup mechanisms:

- Backup Data Files
- Value Files
- Linear Record Files
- Cyclic Record Files

N**OTE:**
The CommitTransaction is typically the last command of a transaction before the ISO 14443-4 Deselect command or before proceeding with another application (SelectApplication command).
As logical counter-part of the CommitTransaciton command the AbortTransaction command allows to invalidate changes on files with integrated backup management.

5.3.31 PICC_MF3_ABORTTRANS (0x9F)

The AbortTransaction command allows to invalidate all previous write access on Backup Data Files, Value Files and Record Files within one application.
This is useful to cancel a transaction without the need for re-authentication to the PICC, which would lead to the same functionality.

int DESTransaction(unsigned char Command)

----------**DLL Explanation** ----------

Input parameter:
Command: 0xA7 for AbortTransaction

Return: 0(OK) or Error Code

----------**Protocol Example**----------

Send >>50 00 00 9F CF
Return <<50 00 00 9F CF

The AbortTransaction command invalidates all write access to files with integrated backup mechanisms without changing the authentication status:

- Backup Data Files
- Value Files
- Linear Record Files
- Cyclic Record Files

5.4 ISO14443B Commands

5.4.1 PICCACTIVATE_B (0x41)

int PiccActivateB(unsigned char ucRst_1ms,
unsigned char ucAFI,
unsigned char ucMethod,
unsigned char *pUIDLen,
unsigned char *pUID);

----------DLL Explanation ----------

Input parameters:

ucRst_1ms:
Refer to PiccReset command, if set, the antenna will close for ucRst_1ms(ms) first and then Open
---
ucAFI:
Application Family Identifier
---
ucMethod:
1: probabilistic
Others: slot-marker
---
pUIDLen:
UID length (1 byte)
---
pUID: will be 0x50 as a preamble, and then 4bytes UID and others

Return: 0(OK) or Error Code

Output variables:

*pUIDLen:UID length (1 byte)
*pUID:UID: 0x50 as a preamble, and then 4bytes UID and others Card information

Return:

0(OK) or Error Code

----------Protocol Example----------

>> 50 00 01 41 00 10
<< 50 00 0C 41 50 BB EE 44 11 11 22 33 11 77 83 C3 6B (UID: BB EE 44 11)

5.5 ISO15693 Commands

NOTE: If you are familiar with the 15693 card, you can use the 0x2E command to operate the card.

5.5.1 I2_INVENTORY (0xA1)

```
int ISO15693_Inventory(
        unsigned char flags,
        unsigned char AFI,
        unsigned char masklengh,
        unsigned char *uid,
        unsigned short *resplen,
        unsigned char *resp);
```

----------DLL Explanation ----------

flags:	default is 0x26
AFI:	Your card AFI, or set to default 0x00
masklengh:	Known card serial number length of bit, default is 0x00
* uid:	Known card serial number according to masklengh If masklengh==0, * uid does not need
*resplen:	Responded data length (mostly 8 bytes if one card only)
*resp	Responded data(that is card UID,8 Bytes length each)

---------- Function Return ----------

Return: 0(OK) or Error Code (may be no card)

----------Protocol Example----------

>> 50 00 03 A1 26 00 00 D4
//command code 0xA1
//flag 0x26, 1 slot inventory
//with AFI appended 0x00, this will request all card in Range
//mask length 0x00, without UID
<<F0 00 01 A1 01 51 (0 cards)
<<50 00 08 A1 0D 86 58 00 00 02 04 E0 CC (1 cards, UID (0D 86 58 00 00 02 04 E0), 8bytes)

Notes:

This command is used to Get UID of ISO15693 card

Default flags is 0x26， AFI default is 0x00 (to all ISO15693 card)， masklength default is 0x00 (not including known UID)

Return information is one card UID, resplen is data length in 8 bytes in common, *resp is 8 bytes HEX card number.

5.5.2 I2_STAY_QUIET (0xA2)

```
int ISO15693_Stay_Quiet(
        unsigned char flags,
        unsigned char *uid);
```

----------DLL Explanation ----------

flags:	default is 0x22
* uid	

---------- **Function Return** ----------

Return: 0(OK) or Error Code (may be no card)

----------**Protocol Example**----------

Send >> 50 00 09 A2 22 A6 15 56 3C 17 22 02 E0 D7
//flag 0x22
//uid A6 15 56 3C 17 22 02 E0
Return <<50 00 01 A2 00 F3 (OK)

5.5.3 I2_READ_BLOCK (0xA3)

```
int ISO15693_Read_Block(
        unsigned char flags,
        unsigned char blnr,
        unsigned char nbl,
        unsigned char *uid,
        unsigned short *resplen,
        unsigned char *resp);
```

----------**DLL Explanation** ----------

flags:	default is 0x02(access with no UID appended)
blnr:	The Block number you want to read
nbl:	How much block read in one time, default is 0x01
* uid:	Known card serial number 8 bytes is needed (if flags set to 0x22) or NONE (as flags set to 0x02)
*resplen:	Responded data length (mostly 6 bytes if one block only)
*resp	Responded data(6 bytes) The first byte is reading operation status The second byte is block locked status The third to 6th bytes is block data

---------- **Function Return** ----------

Return: 0(OK) or Error Code (may be no card)

----------**Protocol Example**----------

>>50 00 03 A3 02 00 01 F3
//without UID read 1 block fr block0
//command code 0xA3
//flag 0x02
//which block? 0x00
//how much block? 0x01
<<50 00 06 A3 00 00 11 11 11 12 F6
//status 0x00
//block lock status 0x00
//return 4bytes block data 11 11 11 12

Notes:

This command is used to read card data, only one block data (4 byte) can be read only in one time.

Flags default is 0x02 (not including *uid)； blnr is which block data you need to read out, and start from 0, nbl default is 1;

Return info is 6 bytes in common, the first byte is status code, the second byte is block status, and byte 0x02-0x05 is card

data.

5.5.4 I2_WRITE_BLOCK (0xA4)

int ISO15693_Write_Block(
unsigned char flags,
unsigned char blnr,
unsigned char nbl,
unsigned char *uid,
unsigned char *dtw,
unsigned short *resplen,
unsigned char *resp);

----------DLL Explanation ----------

flags:	default is 0x02(access with no UID appended)
blnr:	The Block number you want to write
nbl:	How much block write in one time, default is 0x01
* uid:	Known card serial number 8 bytes is needed (if flags set to 0x22) or NONE (as flags set to 0x02)
* dtw:	Data to write (according to nbl, default is 0x04 bytes)
*resplen:	Responded data length (mostly 0x00 bytes)
*resp	Responded data (may be 0x00 bytes)

---------- Function Return ----------

Return: 0(OK) or Error Code (may be no card)

----------Protocol Example----------

>>50 00 07 A4 02 00 01 11 22 33 44 B4
//without UID write 1 block (4bytes 11 22 33 44) to block0
<<50 00 02 A4 00 00 F6
Notes:

This command is used to write card data into, only one block (4 bytes) can be written in one time

Flags default is 0x02 (not including *uid); blnr is the data of which block to be written into(* dtw), blnr starts from 0; nbl default is 1; * dtw is the 4 byte HEX data to be written into.

Return info is not including data in common, no need to deal with, only need to check the parameter's operation result, 0 means succeed.

5.5.5 I2_LOCK_BLOCK (0xA5)

int ISO15693_Lock_Block(
unsigned char flags,
unsigned char blnr,
unsigned char *uid,
unsigned short *resplen,
unsigned char *resp);

----------DLL Explanation ----------

flags:	default is 0x02(access with no UID appended)
blnr:	The Block number you want to write

* uid: Known card serial number, 8 bytes is needed (if flags set to 0x22) or NONE (as flags set to 0x02)
*resplen: Responded data length (mostly 0x00 bytes)
*resp: Responded data (may be 0x00 bytes)

---------- **Function Return** ----------

Return: 0(OK) or Error Code (may be no card)

----------**Protocol Example**----------

Send >>50 00 02 A5 02 00 F5 //without UID lock block0
Return <<50 00 01 A5 00 F4

Notes:

This command is used to lock block data, which cannot be rewritten after locked.

Flags default is 0x02 (not including *uid); blnr is the block data to be locked (* dtw), blnr starts from 0;

In common, the first byte returned resp[0] is status code, the second byte resp[1] is block status, commonly is 0x00 and 0x00;

Block status is not equal to 0x00, means this block might be locked already

Parameter operation result, 0 stands for success

5.5.6 I2_SELECT (0xA6)

```
int ISO15693_Select(
        unsigned char flags,
        unsigned char *uid);
```

----------**DLL Explanation** ----------

flags: default is 0x22
* uid:

---------- **Function Return** ----------

Return: 0(OK) or Error Code (may be no card)

----------**Protocol Example**----------

Send >> 50 00 09 A6 22 A6 15 56 3C 17 22 02 E0 D3
//flag 0x22
//uid A6 15 56 3C 17 22 02 E0
Return <<50 00 01 A6 00 F7 (OK)

5.5.7 I2_RESET_TO_READY (0xA7)

```
int ISO15693_Reset_to_Ready(
        unsigned char flags,
        unsigned char *uid);
```

----------**DLL Explanation** ----------

flags: default is 0x22
* uid:

---------- **Function Return** ----------

Return: 0(OK) or Error Code (may be no card)

----------**Protocol Example**----------

Send >> 50 00 09 A7 22 A6 15 56 3C 17 22 02 E0 D2
//flag 0x22
//uid A6 15 56 3C 17 22 02 E0
Return <<50 00 01 A7 00 F6 (OK)

5.5.8 I2_WRITE_AFI (0xA8)

```
int ISO15693_Write_AFI(
        unsigned char flags,
        unsigned char AFI,
        unsigned char *uid,
        unsigned short *resplen,
        unsigned char *resp);
```

----------**DLL Explanation** ----------

flags:	default is 0x02(access with no UID appended)
AFI:	The AFI
* uid:	Known card serial number 8 bytes is needed (if flags set to 0x22) or NONE (as flags set to 0x02)
*resplen:	Responded data length (mostly 0x00 bytes)
*resp	Responded data (may be 0x00 bytes)

---------- **Function Return** ----------

Return: 0(OK) or Error Code (may be no card)

----------**Protocol Example**----------

>>50 00 02 A8 02 07 FF //without UID write AFI of 0x07

<<50 00 01 A8 01 F8

Notes:

This command is used to write AFI

In common, the first byte resp[0] returned is status code, the second byte resp[1] is AFI status, commonly is 0x00 and 0x00;

Block status not equal to 0x00, means this AFI might be locked already

Parameter operation result, 0 stands succeed

It's available to use command of ***I2_GET_SYSTEM_INFO*** to check writing result.

5.5.9 I2_LOCK_AFI (0xA9)

```
int ISO15693_Lock_AFI(
        unsigned char flags,
        unsigned char *uid,
        unsigned short *resplen,
        unsigned char *resp);
```

----------**DLL Explanation** ----------

flags:	default is 0x02(access with no UID appended)
* uid:	Known card serial number
	8 bytes is needed (if flags set to 0x22) or NONE (as flags set to 0x02)
*resplen:	Responded data length (mostly 0x00 bytes)
*resp	Responded data (may be 0x00 bytes)

---------- **Function Return** ----------

Return: 0(OK) or Error Code (may be no card)

----------**Protocol Example**----------

>>50 00 01 A9 02 FA //without UID
<<50 00 01 A9 01 F9

Notes:

This command is used to lock AFI, please note that once it be locked, which cannot be rewritten ;

Commonly the first byte resp[0] returned is status code, the second byte resp[1] is AFI status, common is 0x00 and 0x00;

Block status not equal to 0x00, means this AFI might be locked already;

Parameter operation result, 0 stands succeed.

5.5.10 I2_WRITE_DSFID (0xAA)

```
Int ISO15693_Write_DSFID(
        unsigned char flags,
        unsigned char DSFID,
        unsigned char *uid,
        unsigned short *resplen,
        unsigned char *resp);
```

----------**DLL Explanation** ----------

flags:	default is 0x02(access with no UID appended)
DSFID:	The DSFID
* uid:	Known card serial number
	8 bytes is needed (if flags set to 0x22) or NONE (as flags set to 0x02)
*resplen:	Responded data length (mostly 0x00 bytes)
*resp	Responded data (may be 0x00 bytes)

---------- **Function Return** ----------

Return: 0(OK) or Error Code (may be no card)

----------**Protocol Example**----------

>>50 00 02 AA 02 07 FD

//command code is 0xAA

//flag is 0x02

//without UID write DSFID of 0x07

<<50 00 01 AA 01 FA

Notes:

This command is used to write DSFID

Commonly the first byte resp[0] is status code, the second byte resp[1] is DSFID status, commonly is 0x00 and 0x00;

Block status not equal to 0x00, means this DSFID might be locked already;

Parameter operation result, 0 stands succeed ;

It's available to use command of ***I2_GET_SYSTEM_INFO*** to check writing result.

5.5.11 I2_LOCK_DSFID (0xAB)

```
int ISO15693_Lock_DSFID(
        unsigned char flags,
        unsigned char *uid,
        unsigned short *resplen,
        unsigned char *resp);
```

----------DLL Explanation ----------

flags:	default is 0x02(access with no UID appended)
* uid:	Known card serial number 8 bytes is needed (if flags set to 0x22) or NONE (as flags set to 0x02)
*resplen:	Responded data length (mostly 0x00 bytes)
*resp	Responded data (may be 0x00 bytes)

---------- Function Return ----------

Return: 0(OK) or Error Code (may be no card)

----------Protocol Example----------

>>50 00 01 AB 02 F8 //without UID

<<50 00 01 AB 01 FB

Notes:

This command is used to lock DSFID, please note that once it be locked, which cannot be rewritten again;

Commonly the first byte resp[0] return is status code, the second byte resp[1] is DSFID status, commonly is 0x00 and 0x00;

Block status not equal to 0x00, means this DSFID might be locked already;

Parameter operation result, 0 stands succeed.

5.5.12 I2_GET_SYSTEM_INFO (0xAC)

```
int ISO15693_Get_SysInfor(
        unsigned char flags,
        unsigned char *uid,
        unsigned short *resplen,
        unsigned char *resp);
```

----------DLL Explanation ----------

flags:	default is 0x02(access with no UID appended)
* uid:	Known card serial number
	8 bytes is needed (if flags set to 0x22) or NONE (as flags set to 0x02)
*resplen:	Responded data length
*resp	Responded data

---------- Function Return ----------

Return: 0(OK) or Error Code (may be no card)

----------Protocol Example----------

>>50 00 01 AC 02 FF //without UID
<<50 00 10 AC 00 00 0F A6 15 56 3C 17 22 02 E0 00 00 3F 03 22 F3
//00 00 0F- card status and information
//A6 15 56 3C 17 22 02 E0 - UID
//00 - DSFID
//00 - AFI
//3F 03 - Card Memory size
//22 - IC infor and product code

Notes:

To read card information, please refer to card's datasheet for details, which the info will be including in order:

UID - 8 bytes

DSFID - 1 byte

AFI - 1 byte

Card Memory size -2 bytes

IC infor - 1 byte

5.5.13 I2_GET_MultipleBlockSecurityStatus (0xAD)

int ISO15693_GetMultipleBlockSecurityStatus (
unsigned char flags,
unsigned char FirstBlock_addr,
unsigned char Num_Blocks,
unsigned char *uid,
unsigned short *resplen,
unsigned char *resp);

----------DLL Explanation ----------

flags:	default is 0x02(access with no UID appended)
FirstBlock_addr:	the address of start block
Num_Blocks:	numbers of blocks
* uid:	Known card serial number
	8 bytes is needed (if flags set to 0x22) or NONE (as flags set to 0x02)
*resplen:	Responded data length
	----
*resp	Responded data

---------- **Function Return** ----------

Return: 0(OK) or Error Code (may be no card)

----------**Protocol Example**----------

Send >>50 00 03 AD 02 00 10 EC

//02- Flag

//00- start block, block0

//10- numbers of blocks, 16 blocks

Return <<50 00 12 AD 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 EE

// 00-RFU

// 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00- Lock status from Block0 to Block15

Among them 00 stands for unlocked, 01 means locked

5.6 ISO7816 commands

5.6.1 ICCPOWERUP_ISO (0x61)

int IccPowerUp(
unsigned char usCardSlot,
unsigned char *pRecLen,
unsigned char *pRcvBuf)

----------DLL Explanation ----------

usCardSlot: Which Slot to be operated?
01 = The 1st SAM Slot
02 = The 2nd SAM Slot
03 = The 3rd SAM Slot
04 = The 4th SAM Slot
* pRecLen: Length of the ATR from the card
* pRcvBuf: ATR from the card

---------- Function Return ----------

Return: 0(OK) or Error Code

----------Protocol Example----------

>>50 00 01 61 01 31(power on The 1st SAM Slot)

<<50 00 16 61 3B 6D 00 00 00 AA 60 03 90 00 33 20 09 60 53 A1 BD 22 00 00 00 00 3F

5.6.2 ICCPOWEROFF (0x64)

int IccPowerDn(unsigned char usCardSlot)

----------DLL Explanation ----------

usCardSlot: Which Slot to be operated?
01 = The 1st SAM Slot
02 = The 2nd SAM Slot
03 = The 3rd SAM Slot
04 = The 4th SAM Slot

---------- Function Return ----------

Return: 0(OK) or Error Code

----------Protocol Example----------

>>50 00 01 64 01 34(power off The 1st SAM Slot)

<<50 00 00 64 34

5.6.3 ICCAPDU (0x65)

int IccAPDU(
unsigned char usCardSlot,
unsigned int usSendLen,
unsigned char *pSendBuf,
unsigned int *pRcvLen,
unsigned char *pRcvBuf)

----------**DLL Explanation** ----------

usCardSlot: Which Slot to be operated?
01 = The 1st SAM Slot
02 = The 2nd SAM Slot
03 = The 3rd SAM Slot
04 = The 4th SAM Slot
usSendLen: length of data to be sent to the card
*pSendBuf: Data to be sent to the card
pRcvLen: length of data received from the card
* pRcvBuf: Data received from the card

---------- **Function Return** ----------

Return: 0(OK) or Error Code

----------**Protocol Example**----------

>>50 00 06 65 01 00 84 00 00 08 BE (APDU: 00 84 00 00 08)

<<50 00 0A 65 11 22 33 44 55 66 77 88 90 00 27

5.6.4 ICCCHECK_PRES (0x68)

int IccCheck_Pres (unsigned char usCardSlot, unsigned char usStatus)

----------**DLL Explanation** ----------

usCardSlot: Which Slot to be operated?
01 = The 1st SAM Slot
02 = The 2nd SAM Slot
03 = The 3rd SAM Slot
04 = The 4th SAM Slot
usStatus: 00 no card
01 have card

---------- **Function Return** ----------

Return: 0(OK) or Error Code

----------**Protocol Example**----------

>>50 00 01 68 01 38

<<50 00 01 68 00 39

5.6.5 ICCSETBAUDRATE (0x6B)

int IccSetInitBaudrate(unsigned char usCardSlot, unsigned char ucRates)

---------- **Explanation** ----------

This command is used for changing the communication clock frequency.

This command should be set before power up the card.

----------**DLL Explanation** ----------

usCardSlot: Which Slot to be operated?
01 = The 1st SAM Slot
02 = The 2nd SAM Slot

03 = The 3rd SAM Slot
04 = The 4th SAM Slot

ucRates: (Parameter only)

Parameter	Baud rate (Baud)
04	9600
03	19200
02	38400
01	57600
00	115200

---------- **Function Return** ----------

Return: 0(OK) or Error Code

----------**Protocol Example**----------

>>50 00 02 6B 01 02 3A (set SAM slot1 init baudrate = 38400)

>>50 00 02 6B 01 04 3C (set SAM slot1 init baudrate = 9600)

<<50 00 00 6B 3B

Note: parameter set is not saved if power down. The default baudrate is 9600 bps after power on.

5.7 ISO18000-3M3 Commands

5.7.1 General Definitions

Memory Banks

```
#define PHAL_I18000P3M3_MEMBANK_RESERVED    0x00U   /** < Reserved Memory Bank. */
#define PHAL_I18000P3M3_MEMBANK_UII         0x01U   /** < UII Memory Bank. */
#define PHAL_I18000P3M3_MEMBANK_TID         0x02U   /** < TID Memory Bank. */
#define PHAL_I18000P3M3_MEMBANK_USER        0x03U   /** < User Memory Bank. */
```

5.7.2 ISO18000P3M3_INVENTORY (0xB1)

Notes

```
Status_t ISO18000P3M3_Activate(
      uint8_t *rebInforLen,
      uint8_t *rebInfor,
      uint8_t *pbM,uint8_t *pbDr
);
```

Telegram Example

Command from PC/PLC to RFID: 50 00 00 B1 E1

Reply form RFID to PC/PLC: 50 00 0E B1 03 01 00 00 00 00 00 00 48 04 27 C4 5D 5B 44

The Bytes in Detail:

50	= Start of telegram
00 0E	= 14 Bytes of payload between command code and CRC
B1	= Command code
03	= Modulation
01	= Link frequency
00 00 00 00	
00 00 48 04	
27 C4 5D 5B	= 12 Bytes of EPC
44	= CRC

5.7.3 ISO18000P3M3_ACK (0xB2)

Acknowledge a single tag.

Notes

```
Status_t ISO18000p3m3_Ack(
      uint8_t * pRxBuffer,    /**< [Out] Pointer to Tag data and, if required, PacketCRC. */
      uint16_t * pRxLength    /**< [Out] Tag response length in bits. */
);
```

Telegram Example

Command from PC/PLC to RFID: 50 00 00 B2 E2

Reply form RFID to PC/PLC: 50 00 0E B2 30 00 00 00 00 00 00 00 48 04 27 C4 5D 5B 75

The Bytes in Detail:

50	= Start of telegram
00 0E0E	= 14 Bytes of payload between command code and CRC
B2	= Command code
30 00	= Protocol Control (PC)

00 00 00 00
00 00 48 04
27 C4 5D 5B = 12 Bytes of EPC
75 = CRC

5.7.4 ISO18000P3M3_REQRN (0xB3)

Instruct a tag to loadmodulate a new RN16 or Handle.

Notes

```
Status_t ISO18000p3m3_ReqRn(
      uint8_t ** pRxBuffer   /**< [Out] New RN16 or handle. */
);
```

Telegram Example

Command from PC/PLC to RFID: 50 00 00 B3 E3
Reply form RFID to PC/PLC: 50 00 02 B3 00 00 44
The Bytes in Detail:

50 = Start of telegram
00 02 = 2 Bytes of payload between command code and CRC
B3 = Command code
00 00 = New RN16 or handle
44 = CRC

5.7.5 ISO18000P3M3_READ(0xB4)

Description

Read part or all of a tag Reserved, UII, TID, or User memory.
bWordPtrLength depends on the TAG memory size. For TAGs with 8 bits memory, bWordPtrLength should be always '0'. If we make 'bWordPtrLength' =1 (16bits) or higher for 8 bits memory TAGs then this function returns MEMORY_OVERRUN error.

Notes

```
Status_t ISO18000p3m3_Read(
      uint8_t bMemBank,        /**< [In] Memory bank where the read shall be performed. */
      uint8_t * pWordPtr,      /**< [In] Starting read address. */
      uint8_t bWordPtrLength,  /**< [In] Length of the pointer in bytes;
                                    0 -> 1byte, 1 -> 2bytes, 2 -> 3bytes or 3 -> 4bytes. */
      uint8_t bWordCount,      /**< [In] Number of bytes to read. */
      uint8_t * pRxBuffer,     /**< [Out] Header and requested memory words. */
      uint16_t * pRxLength     /**< [Out] Number of received bits. */
);
```

Telegram Example 1

Command from PC/PLC to RFID: 50 00 05 B4 03 00 00 00 02 E0 //USER(03) 2*2 bytes

The Bytes in Detail:

50 = Start of telegram
00 05 = 5 Bytes of payload between command code an CRC
B4 = Command code
03 = Memory bank
00 00 = Start address in blocks, LSB first!, only even numbers!
00 02 = Number of blocks to read, 1 block = 2 Bytes
E0 = CRC

Reply form RFID to PC/PLC: 50 00 04 B4 00 00 00 00 E0

The Bytes in Detail:

50	= Start of telegram
00 04	= 4 Bytes of payload between command code and CRC
B4	= Command code
00 00 00 00	= 4 Bytes of data
E0	= CRC

Telegram Example 2

Command from PC/PLC to RFID: 50 00 05 B4 02 00 00 00 04 E7 //TID(02) 4*2 bytes
Reply form RFID to PC/PLC: 50 00 08 B4 E2 00 68 03 00 00 48 04 29

Telegram Example 3

Command from PC/PLC to RFID: 50 00 05 B4 01 00 00 00 08 E8 //EPC(01) 0x08*2 bytes = 16bytes
Reply form RFID to PC/PLC: 50 00 10 B4 EC D0 30 00 00 00 00 00 00 00 48 04 27 C4 5D 5B 51

Telegram Example 4

Command from PC/PLC to RFID: 50 00 05 B4 02 02 00 00 04 E5 Read 4 blocks, start at block 2
Reply form RFID to PC/PLC: 50 00 08 B4 00 00 48 01 38 F2 55 2D 17

Telegram Example 5

Command from PC/PLC to RFID: 50 00 05 B4 02 04 00 00 02 E5 Read 2 blocks, start at block 4
Reply form RFID to PC/PLC: 50 00 04 B4 38 F2 55 2D 52

5.7.6 ISO18000P3M3_WRITE(0xB5)

Description

Write a word in a tag Reserved, UII, TID, or User memory.

bWordPtrLength depends on the TAG memory size. For TAGs with 8 bits memory, bWordPtrLength should be always '0'. If we make 'bWordPtrLength' =1 (16bits) or higher for 8 bits memory TAGs then this function returns MEMORY_OVERRUN error. This is an expected behaviour.

bOption can be one of: #PHAL_I18000P3M3_AC_NO_COVER_CODING
#PHAL_I18000P3M3_AC_USE_COVER_CODING

Notes

```
Status_t ISO18000p3m3_Write(
      uint8_t bOption,         /**< [In] Option parameter. */
      uint8_t bMemBank,        /**< [In] Memory bank where the write shall be performed. */
      uint8_t * pWordPtr,      /**< [In] Starting write address. */
      uint8_t bWordPtrLength, /**< [In] Length of the pointer in bytes;
                                    0 -> 1byte, 1 -> 2bytes, 2 -> 3bytes or 3 -> 4bytes. */
uint8_t * pData          /**< [In] Word to write; uint8_t[2]. */
);
```

Telegram Example

Command from PC/PLC to RFID: 50 00 07 B5 01 03 00 00 00 11 22 D3 //USER(03) 1*2 bytes
The Bytes in Detail:

50	= Start of Telegram
00 07	= 7 Bytes of payload between command code and CRC

B5 = Command code
01 = Option Byte
03 = Memory bank
00 00 = Start address in blocks, LSB first!
00 = Number of blocks to write, 0 = 1, 1 = 2 (not supported by I-Code ILT-M)
11 22 = Data
D3 = CRC

Reply form RFID to PC/PLC: 50 00 00 B5 E5

Notes

The I-Code ILT-M supports to write only 1 block at once using this command.

```
#define PHAL_I18000P3M3_AC_NO_COVER_CODING  0x00U   /**< Use cover coding
                                                         to diversify passwords. */
#define PHAL_I18000P3M3_AC_USE_COVER_CODING 0x01U   /**< Do not use cover coding,
                                                         send plain passwords. */
```

5.7.7 ISO18000P3M3_KILL(0xB6)

Description

Render a tag killed or recommissioned as appropriate.

bOption can be one of: #PHAL_I18000P3M3_AC_NO_COVER_CODING
#PHAL_I18000P3M3_AC_USE_COVER_CODING

Notes

```
Status_t ISO18000p3m3_Kill(
      uint8_t bOption,     /**< [In] Option parameter. */
      uint8_t * pPassword, /**< [In] Full kill password; uint8_t[4] */
      uint8_t bRecom       /**< [In] Recommissioning bits. */
);
```

Telegram Example

Command from PC/PLC to RFID: 50 00 04 B6 01 00 00 00 00 00 E2 //send plain passwords(01) 4 bytes pPassword
The Bytes in Detail:

50 = Start of telegram
00 04 = 4 Bytes of payload between command code and CRC
B6 = Command code
01 = Option parameter
00 00 00 00 = Kill password
00 = Recommissioning bits
E2 = CRC

Reply form RFID to PC/PLC: 50 00 00 B6 E6

Notes

```
#define PHAL_I18000P3M3_AC_NO_COVER_CODING  0x00U   /**< Use cover coding
                                                         to diversify passwords. */
#define PHAL_I18000P3M3_AC_USE_COVER_CODING 0x01U   /**< Do not use cover coding,
                                                         send plain passwords. */
```

5.7.8 ISO18000P3M3_LOCK(0xB7)

Description

Lock or Permalock individual passwords and memory banks.

Notes

```
Status_t ISO18000p3m3_Lock (
      uint8_t * pMask,     /**< [In] 10bit Action Field Mask; uint8_t[2]. */
      uint8_t * pAction    /**< [In] 10bit Action Field; uint8_t[2]. */
);
```

Telegram Example

Command from PC/PLC to RFID: 50 00 04 B7 00 00 00 00 E3

The Bytes in Detail:

50	= Start of telegram
00 04	= 4 Bytes of payload between command code and CRC
B7	= Command code
00 00	= pMask
00 00	= pAction
E3	= CRC

Reply form RFID to PC/PLC: 50 00 00 B7 E7

5.7.9 ISO18000P3M3_ACCESS (0xB8)

Cause a tag with a non-zero-valued access password to transition from the open to the secured state.

bOption can be one of: #PHAL_I18000P3M3_AC_NO_COVER_CODING
#PHAL_I18000P3M3_AC_USE_COVER_CODING

Notes

```
Status_t ISO18000p3m3_Access(
      uint8_t bOption,       /**< [In] Option parameter. */
      uint8_t * pPassword    /**< [In] Full access password; uint8_t[4] */
);
```

Telegram Example

Command from PC/PLC to RFID: 50 00 05 B8 01 00 00 00 00 EC

The Bytes in Detail:

50	= Start of telegram
00 05	= 5 Bytes of payload between command code and CRC
B8	= Command code
01	= Option parameter
00 00 00 00	= Password
EC	= CRC

Reply form RFID to PC/PLC: 50 00 00 B8 E8

5.7.10 ISO18000P3M3_BLOCKWRITE (0xB9)

Write multiple words in a tag Reserved, UII, TID, or User memory.

Return value: Status code #PH_ERR_SUCCESS = Operation successful.
Other Depending on implementation and underlaying component.

Notes

The I-Code ILT-M supports to write only 2 block at once using this command.

```
Status_t ISO18000p3m3_BlockWrite(
      uint8_t bMemBank,          /**< [In] Memory bank where the write shall be performed. */
      uint8_t * pWordPtr,        /**< [In] Starting write address. */
      uint8_t bWordPtrLength,    /**< [In] Length of the pointer in bytes; 0,1,2,3 */
      uint8_t bWordCount,        /**< [In] Number of blocks to write. */
      uint8_t * pData            /**< [In] Words to write; uint8_t[2U * \c bWordCount]. */
);
```

Telegram Example

Command from PC/PLC to RFID: 50 00 09 B9 03 00 00 00 02 11 22 33 44 A5 //USER(03) 2*2 bytes

The Bytes in Detail:

50	= Start of telegram
00 09	= 9 Bytes of payload between command code and CRC
B9	= Command code
03	= Memory bank, 0x03 = USER
00 00	= Start address
00	= Length of pointer
02	= Number of blocks
11 22 33 44	= Data to be written, 4 Bytes = 2 blocks
A5	= CRC

Reply form RFID to PC/PLC: 50 00 00 B9 E9

5.7.11 ISO18000P3M3_BLOCKERASE (0xBA)

Erase multiple words in a tag Reserved, UII, TID, or User memory.

Notes

```
Status_t ISO18000p3m3_BlockErase(
      uint8_t bMemBank,          /**< [In] Memory bank where the erase shall be performed. */
      uint8_t * pWordPtr,        /**< [In] Starting erase address. */
      uint8_t bWordPtrLength,    /**< [In] Length of the pointer in bytes; 0,1,2,3. */
      uint8_t bWordCount         /**< [In] Number of blocks to erase. */
);
```

Telegram Example

Command from PC/PLC to RFID: 50 00 05 BA 03 00 00 00 04 E8 //USER(03)

The Bytes in Detail:

50	= Start of telegram
00 05	= 5 Bytes of payload between command code and CRC
BA	= Command code
03	= Memory bank, 0x03 = USER
00 00	= Start address

00 = Length of the pointer
04 = Number of blocks to erase
E8 = CRC

Reply form RFID to PC/PLC: 50 00 00 BA EA

5.7.12 ISO18000P3M3_BLOCKPERMALOCK (0xBB)

Erase multiple words in a tag Reserved, UII, TID, or User memory.

Notes

```
Status_t ISO18000p3m3_BlockPermaLock(
      uint8_t bRFU,               /**< [In] RFU, shall be set to \c 0. */
      uint8_t bReadLock,          /**< [In] Whether the permalock states shall be retrieved
                                       (\c 0) or the blocks shall be permalocked (\c 1). */
      uint8_t bMemBank,           /**< [In] Memory bank where the erase shall be performed. */
      uint8_t * pBlockPtr,        /**< [In] Starting erase adress. */
      uint8_t bBlockPtrLength,    /**< [In] Length of the pointer in bytes; 0,1,2,3. */
      uint8_t bBlockRange,        /**< [In] Mask range, specified in units of 16 blocks. */
      uint8_t * pMask,            /**< [In] Specifies which memory blocks a tag permalocks;
                                       uint8_t[2U * \c bBlockRange]
                                       Ignored if \c bReadLock is \c 0 */
      uint8_t *pRxBuffer,         /**< [Out] Header and Permalock bits
                                       if \c bReadLock is \c 0 or NULL otherwise. */
      uint16_t * pRxLength        /**< [Out] Number of received bytes
                                       if \c bReadLock is \c 0. */
);
```

Telegram Example

Command from PC/PLC to RFID: 50 00 09 BB 00 00 00 00 00 00 02 00 00 EE

The Bytes in Detail:

50 = Start of telegram
00 09 = 9 Bytes of payload between command code and CRC
B8 = Command code
01 = Option parameter
00 00 00 00 = Password
EC = CRC

Reply form RFID to PC/PLC: 50 00 0x BB yy EB

5.7.13 ISO18000P3M3_SETHANDLE (0xBC)

Set the Handle into the internal data structure.

Notes

```
Status_t ISO18000p3m3_SetHandle(
      uint8_t* pHandle            /**< [In] Handle to the Card. */
);
```

Telegram Example

Command from PC/PLC to RFID: 50 00 02 BC 00 00 EE

The Bytes in Detail:

50	= Start of telegram
00 02	= 2 Bytes of payload between command code and CRC
BC	= Command code
00 00	= Handle word
EC	= CRC

Reply form RFID to PC/PLC: 50 00 00 BC 44

6 Com Operation

6.1 Check Data

int LRC(unsigned short int len, char *buff)

---------- **Explanation** ----------

This command is used for checking if the data is accord with the protocol

This command can be use as a LRC tool, the last byte of the data will automatically changed and set to accord with the protocol

----------**DLL Explanation** ----------

len:	Length of the data to be LRC
buff:	Data to be LRC

---------- **Function Return** ----------

Return: 0(accordant) or 1(not accordant but will set to accordant)

6.2 Open Port

int open(int port,long baudrate)

---------- **Explanation** ----------

Com should be open before and command sending

----------**DLL Explanation** ----------

port:	UART port number
	1
	2
	3
	4
	5
	…
baudrate:	9600
	19200
	38400
	57600
	115200

---------- **Function Return** ----------

Return: 0(OK) or Error Code

6.3 Close Port

int close(void)

---------- **Explanation** ----------

No need any parameter, will close the com which have been open

6.4 Set Baudrate

int baud(long baudrate);

----------**DLL Explanation** ----------

baudrate: 9600
19200
38400
57600
115200

---------- **Function Return** ----------

Return: 0(OK) or Error Code

6.5 Set Timeout

int timeout(int time);

---------- **Explanation** ----------

Timeout between send and receive data from the Com

Recommend to set larger than 15000(1/10ms) or not using

If set to 15000, that is 1500ms

----------**DLL Explanation** ----------

time: Timeout (1/10ms)

---------- **Function Return** ----------

Return: 0(OK) or Error Code