READER-HOST-PROTOCOL

iDTRONIC GmbH · Donnersbergweg 1 · 67059 Ludwigshafen · support@idtronic.de · idtronic-group.com

History of Change:

13.07.2009	R&D	Changed output power to attenuation.	v0.01
28.07.2009	R&D	Added Lock command	v0.02
03.08.2009	R&D	Adapted Return codes and enums	v0.03
10.11.2009	R&D	Added state information and interrupt	v0.04
30.04.2010	R&D	Added sensitivity setting	v0.05
12.05.2010	R&D	Added the status register	v0.06
24.01.2011	R&D	Added GPIO commands	v0.07
09.05.2011	R&D	Added duplex heartbeat description	v0.08
08.12.2011	R&D	Added LBT Params, Added Boot-Up-Finished Interrupt, Added Notifications	v0.09
21.12.2011	R&D	Added Antenna Commands	v0.10
06.09.2012	R&D	Added return code RESULT_PENDING	v0.11
09.11.2012	R&D	Added GPIO-Values-Changed Interrupt	v0.12
30.01.2013	R&D	Added new types of heartbeat	v0.13
12.02.2013	R&D	Corrected some parts	v0.14
26.06.2014	R&D	Corrected some parts	V0.15

iDTRONIC GmbH · Donnersbergweg 1 · 67059 Ludwigshafen · support@idtronic.de · idtronic-group.com

1 Introduction 6

2 Packet Transmission 7

3 Packet Structure 8

4 Return Codes 9

5 Variable Dictionary 10

5.1 Reader Common (01) 10

5.2 RF-Settings (02) 10

5.3 Reader Control (03) 11

5.4 Tag-Mode (04) 11

5.5 GPIO (05) 11

5.6 Antenna (06) 12

5.7 Notifications (10) 12

5.8 Tag Control (50) 12

5.9 Reader-Interrupts (90) 12

6 Detailed Description 13

6.1 Get-Serial-Number (01-01) 13

6.2 Get-Reader-Type (01-02) 13

6.3 Get-Hardware-Revision (01-03) 13

6.4 Get-Software-Revision (01-04) 14

6.5 Get-Bootloader-Revision (01-05) 14

6.6 Get-Current-State (01-07) 15

6.7 Get-Status-Register (01-08) 15

6.8 Get-Antenna-Count (01-10) 15

6.9 Get-Attenuation (02-01) 16

6.10 Get-Frequency (02-02) 16

6.11 Get-Sensitivity (02-03) 17

6.12 Get-LBT-Params (02-03) since v1.07 17
6.13 Set-Attenuation (02-81) 18
6.14 Set-Frequency (02-82) 19
6.15 Set-Sensitivity (02-83) 20
6.16 Set-LBT-Params (02-84) since v1.07 21
6.17 Reboot (03-01) 21
6.18 Set-Heartbeat (03-02) 22
6.19 Set-Antenna-Power (03-03) 23
6.20 Restore-Factory-Settings (03-20) 24
6.21 Save-Settings-Permanent(03-21) 24
6.22 Set-Param (03-30) 25
6.23 Get-Param (03-31) 26
6.24 Set DeviceName (03-32) 27
6.25 Get DeviceName (03-33) 27
6.26 Set Device Location (03-34) 28
6.27 Get Device Location (03-35) 28
6.28 Set-Tag-Mode (04-01) (not implemented yet) 29
6.29 Get Current-Tag-Mode (04-02) (not implemented yet) 30
6.30 Get Tag-Function-List (04-03) (not implemented yet) 30
6.31 Get-GPIO-Caps (05-01) 31
6.32 Get-GPIO-Direction (05-02) 31
6.33 Set-GPIO-Direction (05-03) 32
6.34 Get-GPIO (05-04) 32
6.35 Set-GPIO (05-05) 33
6.36 Clear-GPIO (05-06) 33
6.37 Set Antenna Sequence (06-01) since v1.08 34
6.38 Get Antenna Sequence (06-02) since v1.08 35
6.39 Set Working Antenna (06-03) since v1.08 36
6.40 Get Working Antenna (06-04) since v1.08 36

6.41 Activate Notification (10-01) 37
6.42 Deactivate Notification (10-02) 37
6.43 Get Active Notifications (10-03) 38
6.44 Inventory-Single (50-01) 38
6.45 Inventory-Cyclic (50-02) 40
6.46 Read-From-Tag (50-03) 41
6.47 Write-To-Tag (50-04) 42
6.48 Lock-Tag (50-05) 43
6.49 Kill-Tag (50-06) 44
6.50 Custom-Tag-Command (50-10) 45
6.51 Read-Multiple-From-Tag (50-20) 46
6.52 Heartbeat-Interrupts (90-01) 48
6.53 Inventory-Cyclic-Interrupt (90-02) 48
6.54 State-Changed-Interrupt (90-03) 49
6.55 Status-Reg-Changed-Interrupt (90-04) 49
6.56 Boot-Up Finished Interrupt (90-05) 49
6.57 Notification Interrupt (90-06) 49
6.58 Operation-Result-Interrupt (90-08) 50
6.59 GPIO-Values-Changed-Interrupt (90-09) 50

7 Pending Results 51

8 Data Structures 52

1 Introduction

This document describes the communication protocol that is used for the UHF-RFID OEM, Desktop and Stick Readers. The protocol was designed to work with all these products (active and passive) and independent of which interface is used for the communication. So it is a very small and essential protocol but has the possibility to be enhanced.

In the following chapters the structures and the values of the protocol are described.

2 Packet Transmission

This protocol does not depend on the interface which is used for the communication. So with the use of every interface the data structure is the same.

3 Packet Structure

All exchanged packets follow the same structure as shown below:

Start Bytes	3 Bytes
Command Start Byte	1 Byte
Command	2 Bytes
Length Start Byte	1 Byte
Length	1 Byte
Payload Start Byte	1 Byte
Payload	Variable
Checksum Start Byte	1 Byte
Checksum	1 Byte

Start Byte: The start bytes are used to signalize the start of a new packet.

They are always **0x52**, **0x46**, **0x45** (ASCII: R, F, E).

Command1-2: The two command fields describe which command should be executed. These commands are described in the chapter 5.

Length & Payload: These two fields contain the payload and the length of this packet. The length indicates the count of characters in the payload field. If length is zero, the payload start byte and payload can / must be left out.

Checksum: The checksum is just a simple XOR connection of all data before.

All other start bytes are used to synchronize the protocol and to reduce the probability of the misinterpretation of the byte sequence.

Example: (Save-Settings-Permanent)

52 46 45 01 0321 02 01 03 00 04 cs

start bytes: appear in every message, except the payload start byte (0x03)

commands: always appear in a row of two bytes

length: one byte, max length is 255

payload: includes data or parameter like modes, IDs, return results etc…

cs: checksum, xor conjunction of all preceding bytes of that messages

4 Return Codes

The return codes of all reader functions are listed below:

Name	Value	Description
RFE_RET_SUCCESS	0x00	Everything went fine.
RFE_RET_RESULT_PENDING	0x01	The operation is pending, the result will be sent later on.
RFE_RET_ERR_OP_NOT_SUPPORTED	0x50	Operation is not supported on this reader.
RFE_RET_ERR_UNKOWN_ERR	0x51	Unkown error.
RFE_RET_ERR_ON_EXEC_OP	0x52	The operation could not be executed.
RFE_RET_ERR_COULD_NOT_WRITE	0x53	The reader could not write the value.
RFE_RET_ERR_WRONG_PARAM_COUNT	0x54	The function was called with the wrong parameter count.
RFE_RET_ERR_WRONG_PARAM	0x55	The function was called with the wrong parameter.

The return codes of the Tag Manipulator Interface (TMI) that can be returned when a tag should be manipulated are listed below:

Name	Value	Description
RFE_RET_TMI_TAG_UNREACHABLE	0xA0	The reader could not reach the tag.
RFE_RET_TMI_MEM_OVERRUN	0xA1	The specified memory space is not valid.
RFE_RET_TMI_MEM_LOCKED	0xA2	The specified memory space is locked.
RFE_RET_TMI_INSUFFICIENT_POWER	0xA3	The tag has too less power.
RFE_RET_TMI_WRONG_PASSWORD	0xA4	The specified password is wrong.

This table should be used as an enum with the name RFE_RET_VALUE. This type is used in the following description. It is used as an unsigned char.

5 Variable Dictionary

5.1 Reader Common (01)

Command Byte 1		Command Byte 2		Version
Reader-Common	0x01	Get-Serial Number	0x01	
		Get-Reader Type	0x02	
		Get-Hardware Revision	0x03	
		Get-Software Revision	0x04	
		Get-Bootloader Revision	0x05	
		Get-Current-System	0x06	
		Get-Current-State	0x07	
		Get-Status-Register	0x08	
		Get-Antenna-Count	0x10	1.08

5.2 RF-Settings (02)

Command Byte 1		Command Byte 2		Version
Reader-RF	0x02	Get-Attenuation	0x01	
		Get-Frequency	0x02	
		Get-Sensitivity	0x03	
		Get-LBT-Params	0x04	1.07
		Set-Attenuation	0x81	1.08
		Set-Frequency	0x82	1.08
		Set-Sensitivity	0x83	1.08
		Set-LBT-Params	0x84	1.08

5.3 Reader Control (03)

Command Byte 1		Command Byte 2		Version
Reader-Control	0x03	Reboot	0x01	
		Set-Heart-Beat	0x02	
		Set-Antenna-Power	0x03	
		~~*Set-Attenuation (dBm)*~~	~~*0x04*~~	*Deprecated since 1.08*
		~~*Set-Frequency*~~	~~*0x05*~~	*Deprecated since 1.08*
		~~*Set-Sensitivity*~~	~~*0x06*~~	*Deprecated since 1.08*
		~~*Set-LBT-Params*~~	~~*0x07*~~	*Deprecated since 1.08*
		Restore-Factory-Settings	0x20	
		Save-Settings-Permanent	0x21	
		Set-Param	0x30	
		Get-Param	0x31	
		Set-Device-Name	0x32	1.04
		Get-Device-Name	0x33	1.04
		Set-Device-Location	0x34	1.04
		Get-Device-Location	0x35	1.04

5.4 Tag-Mode (04)

Command Byte 1		Command Byte 2		Version
Reader-Tag-Mode	*0x04*	*Set-Tag-Mode*	*0x01*	*Not yet implemented*
		Get-Current-Tag-Mode	*0x02*	*Not yet implemented*
		Get-Tag-Function-List	*0x03*	*Not yet implemented*

5.5 GPIO (05)

Command Byte 1		Command Byte 2		Version
GPIO-Control	0x05	Get-GPIO-Caps	0x01	
		Get-GPIO-Direction	0x02	
		Set-GPIO-Direction	0x03	
		Get-GPIO	0x04	
		Set-GPIO	0x05	
		Clear-GPIO	0x06	

5.6 Antenna (06)

Command Byte 1		Command Byte 2		Version
Antenna-Control	0x06	Set-Antenna-Sequence	0x01	1.08
		Get-Antenna-Sequence	0x02	1.08
		Set-Working-Antenna	0x03	1.08
		Get-Working-Antenna	0x04	1.08

5.7 Notifications (10)

Command Byte 1		Command Byte 2		Version
Notfications	0x10	Activate-Notifications	0x01	1.07
		Deactivate-Notifications	0x02	1.07
		Get-Active-Notifications	0x03	1.07

5.8 Tag Control (50)

Command Byte 1		Command Byte 2		Version
Tag-Functions	0x50	Inventory-Single	0x01	
		Inventory-Cyclic	0x02	
		Read-From-Tag	0x03	
		Write-To-Tag	0x04	
		Lock-Tag	0x05	
		Kill-Tag	0x06	
		Custom-Tag-Command	0x10	
		Read-Multiple-From-Tag	0x20	

5.9 Reader-Interrupts (90)

Command Byte 1		Command Byte 2		Version
Interrupt	0x90	Heart-Beat-Interrupt	0x01	
		Inventory-Cyclic-Interrupt	0x02	
		State-Changed-Interrupt	0x03	
		Status-Reg-Changed-Interrupt	0x04	
		Boot-Up-Finished	0x05	v1.02
		Notification-Interrupt	0x06	v1.07
		Operation-Result-Interrupt	0x08	
		GPIO-Values-Changed-Interrupt	0x09	v1.16

6 Detailed Description

6.1 Get-Serial-Number (01-01)

This command returns the serial number of the reader.

Return Values: unsigned long **serialNumber**

Example: PC -> Reader: 52 46 45 01 0101 02 00 04 cs

Reader->PC: 52 46 45 01 0101 02 04 03 03000015 04 cs

dataLength = 0x04 -> 4 Bytes

serialNumber = 0x03 0x00 0x00 0x15 -> “03-00-00-15”

6.2 Get-Reader-Type (01-02)

This command returns the reader type of the reader.

Return Values: unsigned long **readerType**

Example: PC -> Reader: 52 46 45 01 0102 02 00 04 cs

Reader->PC: 52 46 45 01 0102 02 04 03 81010101 04 cs

dataLength = 0x04 -> 4 Bytes

readerType = 0x81 0x01 0x01 0x01 -> 81-01-01-01

-> PUR RM1

6.3 Get-Hardware-Revision (01-03)

This command returns the hardware revision of the reader. The version number is split into blocks of 4 bit. One of these blocks represents a decimal character. There are always two characters in front of the point and two after the point. So the first two bytes of the hardware revision are not used.

Return Values: unsigned long **hardwareRevision**

Example: PC -> Reader: 52 46 45 01 0103 02 00 04 cs

Reader->PC: 52 46 45 01 0103 02 04 03 00000115 04 cs

dataLength = 0x04 -> 4 Bytes

hardwareRev = 0x00 0x00 | 0x01 . 0x15 -> “01.15”

6.4 Get-Software-Revision (01-04)

This command returns the software revision of the reader. The version number is split into blocks of 4 bit. One of these blocks represents a decimal character. There are always two characters in front of the point and two after the point. The first two byte of the software reversion define the application version and the second two bytes the version of the used kernel.

Return Values: unsigned long **softwareRevision**

Example: PC -> Reader: 52 46 45 01 0104 02 00 04 cs

Reader->PC: 52 46 45 01 0104 02 04 03 03710107 04 cs

dataLength = 0x04 -> 4 Bytes

softwareRev = 0x03 0x71 | 0x01 0x07

-> App: “03.71” Kernel: “01.07”

6.5 Get-Bootloader-Revision (01-05)

This command returns the bootloader revision of the reader. The version number is split into blocks of 4 bit. One of these blocks represents a decimal character. There are always two characters in front of the point and two after the point. So the first two bytes of the bootloader revision are not used.

Return Values: unsigned long **bootloaderRevision**

Example: PC -> Reader: 52 46 45 01 0105 02 00 04 cs

Reader->PC: 52 46 45 01 0105 02 04 03 00000105 04 cs

dataLength = 0x04 -> 4 Bytes

bootloaderRev = 0x00 0x00 | 0x01 . 0x05 -> “01.05”

6.6 Get-Current-State (01-07)

This command returns the current state of the reader.

Return Values: RFE_RET_STATE **state**

Example: PC -> Reader: 52 46 45 01 0107 02 00 04 cs

Reader->PC: 52 46 45 01 0107 02 01 03 00 04 cs

dataLength = 0x01 -> 1 Bytes

state = 0x00 -> RFE_STATE_IDLE

6.7 Get-Status-Register (01-08)

This command returns the status register of the reader.

Return Values: unsigned long **statusRegister**

Example: PC -> Reader: 52 46 45 01 0108 02 00 04 cs

Reader->PC: 52 46 45 01 0108 02 08 03 00 00 00 00 00 00 00 00 04 cs

dataLength = 0x08 -> 8 Bytes

statusRegister = 0x0000000000000000 -> No Error

6.8 Get-Antenna-Count (01-10)

This command returns the antenna count of the reader.

Return Values: unsigned char **count**

Example: PC -> Reader: 52 46 45 01 0109 02 00 04 cs

Reader->PC: 52 46 45 01 0109 02 01 03 01 04 cs

dataLength = 0x01 -> 1 Bytes

count = 0x01 -> 1 Antenna

6.9 Get-Attenuation (02-01)

This command returns the maximal potential and the current attenuation in dBm.

Parameters: none

Return Values: RFE_RET_VALUE **status**, unsigned short **maxAttenuation**,
unsigned short **currentAttenuation**

Example: PC -> Reader 52 46 45 01 0201 02 00 04 cs

Reader->PC 52 46 45 01 0201 02 07 03 00 000F 000A 04 cs

dataLength = 0x07 -> 7 Bytes

status = SUCCESS

maxAttenuation = 0x000F -> 15 dBm

currentAttenuattion = 0x000A -> 10 dBm

6.10 Get-Frequency (02-02)

This command returns the current frequency and the maximum count of frequencies that can be set.

enum **HOPPING_MODE** (unsigned char)	
STATIC_UP	**0x00**
RANDOM	**0x01**

Parameters: none

Return Values: RFE_RET_VALUE **status**, HOPPING_MODE **mode**, unsigned char **maxFrequencyCount,**
unsigned char **frequencyCount**, unsigned char **frequencys** [frequencyCount] [3]

Example: PC -> Reader 52 46 45 01 0204 02 00 04 cs

Reader->PC

52 46 45 01 0204 02 0A 03 00 01 08 02 0D4094 0D3CAC 04 cs

dataLength = 0x0A -> 10 Bytes

status = 0x00 -> RFE_RET_SUCCESS

mode = 0x01 -> STATIC_DOWN

maxFrequencyCount = 0x08 -> 8 Frequencies

frequencyCount = 0x02 -> 2 Frequencies

frequency1 = 0x0D4094 -> 868500 kHz

frequency2 = 0x0D3CAC -> 867500 kHz

6.11 Get-Sensitivity (02-03)

This command returns the minimal potential and the current sensitivity in dBm.

Parameters: none

Return Values: RFE_RET_VALUE **status**, signed short **maxSensitivity**, signed short **minSensitivity** signed short **currentSensitivity**

Example: PC -> Reader 52 46 45 01 0203 02 00 04 cs

Reader->PC 52 46 45 01 0203 02 07 03 00 FFAC FFCE FFBD 04 cs

dataLength	= 0x07 -> 7 Bytes
status	= SUCCESS
maxSensitivity	= 0xFFAC -> -84 dBm
minSensitivity	= 0xFFCE -> -50 dBm
currentSensitivity	= 0xFFBD -> -67 dBm

6.12 Get-LBT-Params (02-03) since v1.07

This command returns the current set LBT parameters.

Parameters: none

Return Values: RFE_RET_VALUE **status**, unsigned short **listenTime**, unsigned short **idleTime**, unsigned short **maxAllocationTime**, signed short **rssiThreshold**

Example: PC -> Reader 52 46 45 01 0204 02 00 04 cs

Reader->PC 52 46 45 01 0204 02 09 03 00 0001 0000 0190 FFCE 04 cs

dataLength	= 0x09 -> 9 Bytes
status	= SUCCESS
listenTime	= 0x0001 -> 1 ms
idleTime	= 0x0000 -> 0 ms
maxAllocationTime	= 0x0190 -> 400 ms
rssiThreshold	= 0xFFCE -> -50 dBm

6.13 Set-Attenuation (02-81)

This command can be used to set the attenuation of the reader in dBm. The maximal attenuation value can be found in the documentation of the reader or can be read from the reader using the "*Get-Attenuation*" command.

Parameters: unsigned short **value**

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP,
RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM

Example: Set output power to 10 dB

PC -> Reader 52 46 45 01 0281 02 02 03 000A 04 cs

dataLength = 0x02 -> 2 Bytes

value = 0x000A -> 10 dBm

Reader->PC 52 46 45 01 0281 02 01 03 00 04 cs

dataLength = 0x01 -> 1 Bytes

status = 0x00 -> RFE_RET_SUCCESS

6.14 Set-Frequency (02-82)

This command can be used to set frequency at which the reader should operate. It is also possible to specify more than one frequency. This makes only sense with passive reader. The reader will then be hopping from one to another frequency. The mode how the reader should hop through the frequencies can also be specified. These modes are available:

enum **HOPPING_MODE** (unsigned char)	
STATIC_UP	**0x00**
RANDOM	**0x01**

Every frequency is transferred in three bytes. These three bytes specify the frequency in kHz.

Parameters: HOPPING_MODE **mode,** unsigned char **frequencyCount,**
unsigned char **frequencys** [frequencyCount] [3]

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP,
RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM

Example: Set TX frequency to 868.500 and 867.500 MHz

PC -> Reader

52 46 45 01 0282 02 08 03 02 02 0D4094 0D3CAC 04 cs

dataLength = 0x08 -> 8 Bytes
mode = 0x02 -> Random Hopping
frequencyCount = 0x02 -> 2 Frequencies
frequency1 = 0x0D4094 -> 868500 kHz
frequency2 = 0x0D3CAC -> 867500 kHz

Reader->PC 52 46 45 01 0282 02 01 03 00 04 cs

dataLength = 0x01 -> 1 Bytes
status = 0x00 -> RFE_RET_SUCCESS

6.15 Set-Sensitivity (02-83)

This command can be used to set the sensitivity of the reader in dBm. The minimal sensitivity value can be found in the documentation of the reader or can be read from the reader using the "*Get-Sensitivity*" command. Due to reader restrictions not every value can be set for sensitivity. The reader will set the best next value and return this value.

Parameters: signed short **targetValue**

Return Values: RFE_RET_VALUE **status**, signed short **actualValue**

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP,
RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM

Example: Set output power to 10 dB

PC -> Reader 52 46 45 01 0283 02 02 03 FFEC 04 cs

dataLength = 0x02 -> 2 Bytes

targetValue = 0xFFEC -> -20 dBm

Reader->PC 52 46 45 01 0283 02 03 03 00 FFE7 04 cs

dataLength = 0x01 -> 1 Bytes

status = 0x00 -> RFE_RET_SUCCESS

actualValue = 0xFFE7 -> -25 dBm

6.16 Set-LBT-Params (02-84) since v1.07

This command can be used to set the listen before talk parameters of the reader.

Parameters: unsigned short **listenTime**, unsigned short **idleTime**,
unsigned short **maxAllocationTime**, signed short **rssiThreshold**

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP,
RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM

Example: Set settings:

ListenTime:	1 ms
IdleTime:	0 ms
Max AllocationTime:	400 ms
RSSI Threshold:	-50 dBm

PC -> Reader 52 46 45 01 0284 02 08 03 0001 0000 0190 FFCE 04 cs

dataLength	= 0x08 -> 8 Bytes
listenTime	= 0x0001 -> 1 ms
idleTime	= 0x0000 -> 0 ms
maxAllocationTime	= 0x0190 -> 400 ms
rssiThreshold	= 0xFFCE -> -50 dBm

Reader->PC 52 46 45 01 0284 02 01 03 00 04 cs

dataLength	= 0x09 -> 9 Bytes
status	= 0x00 -> RFE_RET_SUCCESS

6.17 Reboot (03-01)

This function can be used to reboot the reader.

Parameters: None

Return Values: None; the reader is rebooted immediately.

Example: PC -> Reader 52 46 45 01 0301 02 00 04 cs

6.18 Set-Heartbeat (03-02)

This command can be used to enable the reader to send a periodic heartbeat. The heartbeat can be turned ON/OFF and an interval for this heartbeat messages can be specified. If no interval is specified the reader takes the interval form the factory settings. This value can be found in the data sheet of the reader.

If the DUPLEX heartbeat is selected, the reader expects the same heartbeat package from the host. If the reader does not get the package from the host in the specified interval, it stops any active scan immediately. The best method to send the heartbeat is to respond to the heartbeat of the reader with the host heartbeat.

If the STATE heartbeat is selected, the reader attaches the current state to each heartbeat package.

enum **HEARTBEAT_SIGNAL** (unsigned char)	
HEARTBEAT _OFF	0x00
HEARTBEAT _ON	0x01
HEARTBEAT_DUPLEX_ON	0x02
HEARTBEAT_STATE_ON	0x03
HEAERTBEAT_DUPLEX_STATE_ON	0x04

Parameters: HeartBeat_Signal **mode**, (unsigned short **interval_in_ms**)

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP,
RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM

Example: Set heartbeat on and send it every 500ms.

PC -> Reader 52 46 45 01 0302 02 03 03 01 01F4 04 cs

dataLength = 0x03 -> 3 Bytes

mode = 0x01 -> HEARTBEAT_ON

interval = 0x01F4 -> 500ms

Reader->PC 52 46 45 01 0302 02 01 03 00 04 cs

dataLength = 0x01 -> 1 Bytes

status = 0x00 -> RFE_RET_SUCCESS

6.19 Set-Antenna-Power (03-03)

This function can be used to set the antenna power on and off. The values are listed below:

enum **ANTENNA_POWER** (unsigned char)	
ANTENNA_OFF	0x00
ANTENNA_ON	0x01

Parameters: AntennaPower **mode**

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP,
RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM

Example: Set antenna power off.

PC -> Reader 52 46 45 01 0303 02 01 03 00 04 cs

dataLength = 0x01 -> 1 Bytes

mode = 0x00 -> ANTENNA_OFF

Reader -> PC 52 46 45 01 0303 02 01 03 00 04 cs

dataLength = 0x01 -> 1 Bytes

status = 0x00 -> RFE_RET_SUCCESS

6.20 Restore-Factory-Settings (03-20)

This function restores the factory settings. All settings are overwritten.

Parameters: None

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP, RFE_RET_ERR_COULD_NOT_WRITE

Example: PC -> Reader 52 46 45 01 0320 02 00 04 cs

Reader -> PC 52 46 45 01 0320 02 01 03 00 04 cs

dataLength = 0x01 -> 1 Byte

status = 0x00 -> RFE_RET_SUCCESS

6.21 Save-Settings-Permanent(03-21)

This function saves all settings permanently to the chip. So the settings are the same after power off.

Parameters: None

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP, RFE_RET_ERR_COULD_NOT_WRITE

Example: PC -> Reader 52 46 45 01 0321 02 00 04 cs

Reader -> PC 52 46 45 01 0321 02 01 03 00 04 cs

dataLength = 0x01 -> 1 Byte

status = 0x00 -> RFE_RET_SUCCESS

6.22 Set-Param (03-30)

Some readers have more settings than it is possible to set with this protocol. So with this function it is possible to set reader specific parameters. The address and the meaning of the fields can be found in the data sheet of the reader. This function should be used very carefully.

Parameters: unsigned short **address**, unsigned char **size,** unsigned char **data** [size]

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP,
RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM

Example: Set the field at 0x0005(address) to the value 0x1785(value).

PC -> Reader 52 46 45 01 0330 02 04 03 0005 02 1785 04 cs

dataLength = 0x04 -> 4 Bytes

address = 0x0005

size = 0x02 -> 2 Byte

data = 0x1785

Reader->PC 52 46 45 01 0330 02 01 03 00 04 cs

dataLength = 0x01 -> 1 Bytes

status = 0x00 -> RFE_RET_SUCCESS

6.23 Get-Param (03-31)

Some readers have more settings than it is possible to set with this protocol. So with this function it is possible to read reader specific parameters. The address and the meaning of the fields can be found in the data sheet of the reader.

Parameters: unsigned short **address**

Return Values: RFE_RET_VALUE **status** , unsigned short **size,** unsigned char **data** [size]

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP,
RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM

Example: Read the field at 0x0005(address).

PC -> Reader 52 46 45 01 0331 02 02 03 0005 04 cs

dataLength = 0x02 -> 2 Bytes

address = 0x0005

Reader->PC 52 46 45 01 0331 02 03 03 00 02 1785 04 cs

dataLength = 0x03 -> 3 Bytes

status = 0x00 -> RFE_RET_SUCCESS

size = 0x02 -> 2 Byte

data = 0x1785

This is only an example!

A valid address and data structure can be found in the data sheet of the reader!

6.24 Set DeviceName (03-32)

With this command a name can be assigned to the reader.

Parameters: char **name** [max 254]

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_WRONG_PARAM_COUNT

Example: Set the name “RFID-Reader”.

PC -> Reader 52 46 45 01 0332 02 0B 03 524649442D526561646572 04 cs

dataLength = 0x0B -> 11 Bytes

name = 0x524649442D526561646572

-> “RFID-Reader”

Reader->PC 52 46 45 01 0332 02 01 03 00 04 cs

dataLength = 0x01 -> 1 Bytes

status = 0x00 -> RFE_RET_SUCCESS

6.25 Get DeviceName (03-33)

With this command a name can be assigned to the reader.

Parameters: None

Return Values: RFE_RET_VALUE **status,** char **name** [max 254]

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_WRONG_PARAM_COUNT

Example: Get the name of the device.

PC -> Reader 52 46 45 01 0333 02 00 04 cs

dataLength = 0x00 -> 0 Bytes

Reader->PC 52 46 45 01 0333 02 0C 03 00 524649442D526561646572 04 cs

dataLength = 0x0C -> 12 Bytes

status = 0x00 -> RFE_RET_SUCCESS

name = 0x524649442D526561646572

-> “RFID-Reader”

6.26 Set Device Location (03-34)

With this command an additional location string can be assigned to the reader.

Parameters: char **location** [max 254]

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_WRONG_PARAM_COUNT

Example: Set the name “POS1”.

PC -> Reader 52 46 45 01 0334 02 04 03 504F5331 04 cs

dataLength = 0x04 -> 4 Bytes

name = 0x504F5331

-> “POS1”

Reader->PC 52 46 45 01 0334 02 01 03 00 04 cs

dataLength = 0x01 -> 1 Bytes

status = 0x00 -> RFE_RET_SUCCESS

6.27 Get Device Location (03-35)

With this command the stored location of the device can be retrieved.

Parameters: None

Return Values: RFE_RET_VALUE **status,** char **name** [max 254]

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_WRONG_PARAM_COUNT

Example: Get the name of the device.

PC -> Reader 52 46 45 01 0335 02 00 04 cs

dataLength = 0x00 -> 0 Bytes

Reader->PC 52 46 45 01 0335 02 0C 03 00 504F5331 04 cs

dataLength = 0x05 -> 5 Bytes

status = 0x00 -> RFE_RET_SUCCESS

name = 0x504F4331

-> “POS1”

6.28 Set-Tag-Mode (04-01) (not implemented yet)

With this function the Tag-Mode can be set. The Tag-Mode says with which type of tags the reader should operate. The tag **modes** are listed below:

enum **TAG_MODE** (unsigned char)			
Passive	*0x00 - 0x7F*		
128 kHz	0x00- 0x1F		
13,56 MHz	0x20 - 0x3F		
868-910 MHz	0x40 - 0x5F		
ISO 18000 6-B		**0x40**	
ISO 18000 6-C / Gen2		**0x41**	
2,4 GHz	0x60 - 0x7F		

Active	*0x80 - 0xFF*		
128 kHz	0x80- 0x9F		
13,56 MHz	0xA0 - 0xBF		
868-910 MHz	0xC0 - 0xDF		
RFE-Active-01		**0xC0**	**Learn Mode**
RFE-Active-02		**0xC1**	**Normal Mode**
2,4 GHz	0xE0 - 0xFF		

Parameters: TAG_MODE **mode**

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP,
RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM

Example: Set tag-mode to Gen2.

PC -> Reader 52 46 45 01 0401 02 01 03 41 04 cs

dataLength = 0x01 -> 1 Bytes

mode = 0x41 -> ISO 18000 6-C / Gen2

Reader -> PC 52 46 45 01 0401 02 01 03 00 04 cs

dataLength = 0x01 -> 1 Bytes

status = 0x00 -> RFE_RET_SUCCESS

6.29 Get Current-Tag-Mode (04-02) (not implemented yet)

This field holds the current tag-mode.

Return Values: unsigned char **tagMode**

Example: PC -> Reader 52 46 45 01 0402 02 00 04 cs

Reader->PC 52 46 45 01 0402 02 01 03 C0 04 cs

dataLength = 0x01 -> 1 Bytes

tagMode = 0xC0 -> RFE-Active-01

6.30 Get Tag-Function-List (04-03) (not implemented yet)

Not every reader can support every function for every tag. So with this function it can figured out, which tag-function the current reader supports for the current selected tag-mode. This function returns a list of the functions that are supported.

Return Values: unsigned char **count**, unsigned char **functions** [count] [3]

Example: PC -> Reader 52 46 45 01 0403 02 00 04 cs

Reader->PC 52 46 45 01 0402 02 05 03 02 5001 5002 04 cs

dataLength = 0x05 -> 5 Bytes

So the supported functions are:

count = 0x02 -> 2 Functions

50 01 -> Inventory

50 02 -> Read-From-Tag

6.31 Get-GPIO-Caps (05-01)

This command returns the capabilities of the reader's GPIO pins. It returns a bitmask, which pins are available, are available as output and are available as input.

Return Values: unsigned long **mask**, unsigned long **output**, unsigned long **input**

Example: PC -> Reader: 52 46 45 01 0501 02 00 04 cs

Reader->PC: 52 46 45 01 0501 02 0C 03 00 00 FF FF 00 00 FF FF 00 00 FF 00 04 cs

dataLength = 0x0C -> 16 Bytes

mask = 0x0000FFFF -> 16 GPIO-pins:

➔ Pin-0 to Pin-15

output = 0x0000FFFF -> 16 Output-pins:

➔ Pin-0 to Pin-15

input = 0x0000FF00 -> 8 GPIO-pins:

➔ Pin-8 to Pin-15

6.32 Get-GPIO-Direction (05-02)

This command returns the current set direction of the pin. If the direction bit for the pin is 1, the direction is output, if the bit is 0, the pin is input.

Return Values: unsigned long **direction**

Example: PC -> Reader: 52 46 45 01 0502 02 00 04 cs

Reader->PC: 52 46 45 01 0502 02 04 03 00 00 0F FF 04 cs

dataLength = 0x04 -> 4 Bytes

direction = 0x00000FFF ->

Refer to the GPIO example before:

- The pins Pin-0 to Pin-11 are configured as output.
- The pins Pin-12 to Pin-15 are configured as input.

6.33 Set-GPIO-Direction (05-03)

This command can be used to change the direction of a pin.

Parameters: unsigned long **directionMask**

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP,
RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM

Example: PC -> Reader: 52 46 45 01 0503 02 04 03 00 00 08 00 04 cs

dataLength = 0x04 -> 4 Bytes

directionMask = 0x00000800 ->

Refer to the GPIO examples before:

- The pin Pin-11 is changed from output to input.

Reader->PC: 52 46 45 01 0503 02 01 03 00 04 cs

dataLength = 0x01-> 1 Bytes

status = 0x00 -> RFE_RET_SUCCESS

6.34 Get-GPIO (05-04)

This command returns the current level of the GPIO-Pins.

Return Values: unsigned long **levelMask**

Example: PC -> Reader: 52 46 45 01 0504 02 00 04 cs

Reader->PC: 52 46 45 01 0504 02 04 03 00 00 98 0F 04 cs

dataLength = 0x04 -> 4 Bytes

directionMask = 0x0000980F ->

Refer to the GPIO examples before:

- The input pins Pin-15, Pin-12 and Pin-11 have a high level.
- The output pins Pin-0 to Pin-3 are set to a high level.

6.35 Set-GPIO (05-05)

This command can be used to set the level of an output pin to high.

Parameters: unsigned long **mask**

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP,
RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM

Example: PC -> Reader: 52 46 45 01 0505 02 04 03 00 00 00 F0 04 cs

dataLength = 0x04 -> 4 Bytes

directionMask = 0x000000F0 ->

Refer to the GPIO examples before:

- The output pins Pin-4 to Pin-7 are set to a high level.

Reader->PC: 52 46 45 01 0505 02 01 03 00 04 cs

dataLength = 0x01-> 1 Bytes

status = 0x00 -> RFE_RET_SUCCESS

6.36 Clear-GPIO (05-06)

This command can be used to set the level of an output pin to low.

Parameters: unsigned long **mask**

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP,
RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM

Example: PC -> Reader: 52 46 45 01 0506 02 04 03 00 00 00 F0 04 cs

dataLength = 0x04 -> 4 Bytes

directionMask = 0x000000F0 ->

Refer to the GPIO examples before:

- The output pins Pin-4 to Pin-7 are set to a low level.

Reader->PC: 52 46 45 01 0506 02 01 03 00 04 cs

dataLength = 0x01-> 1 Bytes

status = 0x00 -> RFE_RET_SUCCESS

6.37 Set Antenna Sequence (06-01) since v1.08

This command sets a sequence for the next cyclic operation. This is done by sending an array of sequence data. These sequence data consist of the selected antenna index and of the time in ms. So the **SequenceStruct** is built up like this:

antennaIndex	unsigned char
activeTime	unsigned long

Parameters: unsigned char **sequenceCount**, SequenceStruct **sequenceData** [sequenceCount]

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM

Example: Use the antennas #1 and #4 for each 2000 ms.

PC -> Reader: 52 46 45 01 0601 02 0B 03 02 01 000007D0 04 000007D0 04 cs

dataLength = 0x0B -> 11 Bytes

sequenceCount = 0x02 -> 2 SequenceStructs

antennaIndex = 0x01 -> Antenna #1

activeTime = 0x000007D0 -> 2000 ms

antennaIndex = 0x04 -> Antenna #4

activeTime = 0x000007D0 -> 2000 ms

Reader->PC: 52 46 45 01 0601 02 01 03 00 04 cs

dataLength = 0x01-> 1 Bytes

status = 0x00 -> RFE_RET_SUCCESS

6.38 Get Antenna Sequence (06-02) since v1.08

This command returns the current set antenna sequence.

Parameters: none

Return Values: RFE_RET_VALUE **status**, unsigned char **sequenceCount**, SequenceStruct **sequenceData** [sequenceCount]

Example: PC -> Reader: 52 46 45 01 0602 02 00 04 cs

PC -> Reader: 52 46 45 01 0602 02 0C 03 02 01 000007D0 04 000007D0 04 cs

dataLength	= 0x0C -> 12 Bytes
status	= 0x00 -> RFE_RET_SUCCESS
sequenceCount	= 0x02 -> 2 SequenceStructs
antennaIndex	= 0x01 -> Antenna #1
activeTime	= 0x000007D0 -> 2000 ms
antennaIndex	= 0x04 -> Antenna #4
activeTime	= 0x000007D0 -> 2000 ms

6.39 Set Working Antenna (06-03) since v1.08

This command selects a single antenna for the next single operations like read, write, lock, …

Parameters: unsigned char **antennaId**

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM

Example: PC -> Reader: 52 46 45 01 0603 02 01 03 02 04 cs

dataLength = 0x01 -> 1 Bytes

antennaId = 0x02 -> Antenna #2

Reader->PC: 52 46 45 01 0603 02 01 03 00 04 cs

dataLength = 0x01-> 1 Bytes

status = 0x00 -> RFE_RET_SUCCESS

6.40 Get Working Antenna (06-04) since v1.08

This command returns the selected working antenna.

Parameters: none

Return Values: unsigned char **antennaId**

Example: PC -> Reader: 52 46 45 01 0604 02 00 04 cs

Reader->PC: 52 46 45 01 0603 02 02 03 00 01 04 cs

dataLength = 0x01-> 2 Bytes

status = 0x00 -> RFE_RET_SUCCESS

antennaId = 0x02 -> Antenna #2

6.41 Activate Notification (10-01)

This command can be used to activate special notifications that are sent automatically by the reader. The possible values for the notifications can be found in the documentation of the reader.

Parameters: unsigned char **id**

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS,
RFE_RET_ERR_WRONG_PARAM_COUNT,
RFE_RET_ERR_WRONG_PARAM

Example: PC -> Reader: 52 46 45 01 1001 02 01 03 01 04 cs
dataLength = 0x01 -> 1 Bytes
id = 0x01 -> e.g. Frequency Change
Reader->PC: 52 46 45 01 1001 02 01 03 00 04 cs
dataLength = 0x01-> 1 Bytes
status = 0x00 -> RFE_RET_SUCCESS

6.42 Deactivate Notification (10-02)

This command can be used to deactivate special notifications that are sent automatically by the reader. The possible values for the notifications can be found in the documentation of the reader.

Parameters: unsigned char **id**

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS,
RFE_RET_ERR_WRONG_PARAM_COUNT,
RFE_RET_ERR_WRONG_PARAM

Example: PC -> Reader: 52 46 45 01 1002 02 01 03 01 04 cs
dataLength = 0x01 -> 1 Bytes
id = 0x01 -> e.g. Frequency Change
Reader->PC: 52 46 45 01 1002 02 01 03 00 04 cs
dataLength = 0x01-> 1 Bytes
status = 0x00 -> RFE_RET_SUCCESS

6.43 Get Active Notifications (10-03)

This command can be used to retrieve the current active notifications. The possible values for the notifications can be found in the documentation of the reader.

Parameters: none

Return Values: unsigned long long **mask**

Example: PC -> Reader: 52 46 45 01 1003 02 00 04 cs

Reader->PC: 52 46 45 01 1003 02 08 03 0000000000000002 04 cs

dataLength = 0x08-> 8 Bytes

mask = 0x0000000000000002

6.44 Inventory-Single (50-01)

This function can be used to make a single inventory round. The reader then returns a list of tags that are in its field. With some interfaces the data length for transmission is limited, so if the return packet would be too large, it is separated into more packets. Therefore two counter are transmitted, one that indicates how many tags were found and one that indicates how many tag id are transmitted in this packet.

The reader can send more additional information then only the tag id. Therefore the structure TagInfo is used. This structure has a variable length that is dependent of the information sent from the reader. The type of additional information is dependent of the used reader. These information and the specific identifier can be found in the documentation of the used reader.

The first byte of the structure indicates the length of the whole structure. After this the tag id is sent. This is splitted in a start byte, a length indicator and the id itself. After that additional information can be added by the reader. In the example below, the RSSI value of the tag is added.

Length-of-TagInfo	**1 Byte**	
StartByte-Tag ID	**1 Byte**	**0x01**
ID-Length	**1 Byte**	
ID	**Length**	
StartByte-RSSI	**1 Byte**	**0x02**
RSSI	**2 Byte**	

Parameters: None

Return Values: RFE_RET_VALUE **status**, unsigned char **idCount**, unsigned char **packetIdCount**, TagInfo **tags** [packetIdCount]

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP, RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM

Example: Do a single inventory. The reader sends the TagInfo with only the TagId.

PC -> Reader 52 46 45 01 5001 02 **00** 04 cs

Reader -> PC

52 46 45 01 5001 02 12 03

00 01 01 0E 01 0C 300833b233330140350500000 04 cs

dataLength	= 0x12
status	= 0x00 -> RFE_RET_SUCCESS
idCount	= 0x01 -> 1 ID
packetIdCount	= 0x01 -> 1 ID

TagInfo 1

Length of TagInfo	= 0x0E -> 14 Bytes
StartByte-TagID	= 0x01
ID Length	= 0x0C -> 12 Bytes
TagID	= 30-08-33-b2-33-33-01-40-35-05-00-00

6.45 Inventory-Cyclic (50-02)

This function can be used to start and stop a cyclic inventory. With a cyclic inventory the reader does inventories autonomous in a specified cycle. The timeout between such cycles can be found in the manual of the reader.

This function can be activated and deactivated:

enum **INVENTORY_MODE** (unsigned char)	
INVENTORY_OFF	**0x00**
INVENTORY_ON	**0x01**

After an activation packet was sent from the PC to the reader, the reader sends an answer with the status of the operation. Afterwards interrupt messages are sent to the PC in a defined cycle.

Parameters: InventoryMode **mode**, [optional ulong **time**]

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP,
RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM

Example: Start cyclic interrupted inventory (**0x01**).

PC -> Reader 52 46 45 01 5002 02 01 03 **01** 04 cs

Reader -> PC 52 46 45 01 5002 02 01 03 00 04 cs

dataLength = 0x01

status = 0x00 -> RFE_RET_SUCCESS

Example: Start cyclic interrupted inventory for 1000 msecs.

PC -> Reader 52 46 45 01 5002 02 05 03 **01** **000003E8** 04 cs

dataLength = 0x05

mode = 0x01 -> Inventory ON

time = 0x000003E8 -> 1000 msecs

Reader -> PC 52 46 45 01 5002 02 01 03 00 04 cs

dataLength = 0x01

status = 0x00 -> RFE_RET_SUCCESS

6.46 Read-From-Tag (50-03)

With this function data can be read from the memory of a tag. The meaning of the data, that can be read, can be found in the manual of the tag. On some tags a memory bank must be specified. The memory banks are specified in the manual of the reader dependent of the type of tag.

Parameters: unsigned char **tagIdCount**, unsigned char **tagId**[tagIdCount]**,**
unsigned char **memoryBank,** unsigned short **startAddress**,
unsigned long **accessPassword,** unsigned char **byteCount**

Return Values: RFE_RET_VALUE **status,** unsigned char **byteCount,** unsigned char **data[byteCount]**

Status Values: RFE_RET_SUCCESS, RFE_RET_RESULT_PENDING, RFE_RET_ERR_ON_EXEC_OP, RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM, Every TMI Return Code

Example: Write 5 byte to the tag 30-08-33-b2-dd-d9-01-40-35-05-00-00 at the memory bank 1 and the start address 0x12:

PC -> Reader

52 46 45 01 5003 02 15 03 0C 300833b2ddd9014035050000 01 0000 00000000 06 04 cs

dataLength	= 0x15
tagIdCount	= 0x0C -> 12 Bytes
tagId	=30-08-33-b2-33-33-01-40-35-05-00-00
memoryBank	= 0x01 -> second bank
startAddress	= 0x0000
accessPassword	= 0x00000000
bytesCount	= 0x06 -> 6 Bytes

Reader -> PC 52 46 45 01 5003 02 08 03 00 06 020023A4884C 04 cs

dataLength	= 0x08
status	= 0x00 -> RFE_RET_SUCCESS
bytesCount	= 0x06 -> 6 Bytes
data	= 0x020023A4884C

6.47 Write-To-Tag (50-04)

With this function data can be written to the memory of a tag. The meaning of the data, that can be written, can be found in the manual of the tag. On some tags a memory bank must be specified. The memory banks are specified in the manual of the reader dependent of the type of tag.

Parameters: unsigned char **tagIdCount**, unsigned char **tagId**[tagIdCount]**,**
unsigned char **memoryBank,** unsigned short **startAddress**,
unsigned long **accessPassword,**
unsigned char **byteCount**, unsigned char **data** [byteCount]

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS, RFE_RET_RESULT_PENDING, RFE_RET_ERR_ON_EXEC_OP, RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM, Every TMI Return Code

Example: Write 5 bytes to the tag 30-08-33-b2-dd-d9-01-40-35-05-00-00 at the memory bank 1 and the start address 0x12:

PC -> Reader

52 46 45 01 5004 02 1B 03 0C 300833b2ddd9014035050000 01 0000 00000000 06 020023A4884C 04 cs

dataLength	= 0x1B
tagIdCount	= 0x0C -> 12 Bytes
tagId	=30-08-33-b2-33-33-01-40-35-05-00-00
memoryBank	= 0x01 -> first bank
startAddress	= 0x0000
accessPassword	= 0x00000000
bytesCount	= 0x06 -> 6 Bytes
data	= 0x020023A4884C

Reader -> PC 52 46 45 01 5004 02 01 03 00 04 cs

dataLength	= 0x01
status	= 0x00 -> RFE_RET_SUCCESS

6.48 Lock-Tag (50-05)

With this function the tag can be locked. The meaning of the mode and the memory space can be found in the documentation of the reader.

Parameters: unsigned char **tagIdCount**, unsigned char **tagId**[tagIdCount]**,**
unsigned char **mode,** unsigned char **mem_space**,
unsigned long **accessPassword,**

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS, RFE_RET_RESULT_PENDING, RFE_RET_ERR_ON_EXEC_OP, RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM, Every TMI Return Code

Example: Lock the tag 30-08-33-b2-dd-d9-01-40-35-05-00-00 with the mode 1 at the memory space 1:

PC -> Reader

52 46 45 01 5005 02 13 03 0C 300833b2ddd9014035050000 01 01 00000000 04 cs

dataLength	= 0x13
tagIdCount	= 0x0C -> 12 Bytes
tagId	=30-08-33-b2-33-33-01-40-35-05-00-00
mode	= 0x01
mem_space	= 0x01
accessPassword	= 0x00000000

Reader -> PC 52 46 45 01 5005 02 01 03 00 04 cs

dataLength	= 0x01
status	= 0x00 -> RFE_RET_SUCCESS

6.49 Kill-Tag (50-06)

With this function the tag can be killed. The meaning of the mode and the memory space can be found in the documentation of the reader.

Parameters: unsigned char **tagIdCount**, unsigned char **tagId**[tagIdCount]**,**
unsigned char **rfuRecom,** unsigned long **accessPassword,**

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS, RFE_RET_RESULT_PENDING, RFE_RET_ERR_ON_EXEC_OP, RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM, Every TMI Return Code

Example: Kill the tag 30-08-33-b2-dd-d9-01-40-35-05-00-00:

PC -> Reader

52 46 45 01 5006 02 12 03 0C 300833b2ddd9014035050000 00 00000000 04 cs

dataLength	= 0x12
tagIdCount	= 0x0C -> 12 Bytes
tagId	=30-08-33-b2-33-33-01-40-35-05-00-00
rfuRecom	= 0x00
accessPassword	= 0x00000000

Reader -> PC 52 46 45 01 5006 02 01 03 00 04 cs

dataLength	= 0x01
status	= 0x00 -> RFE_RET_SUCCESS

6.50 Custom-Tag-Command (50-10)

With this function a custom tag command can be performed. The possible tag commands and the data structure that must be sent to the reader can be found in the documentation of the reader.

Parameters: unsigned char **command**

Return Values: RFE_RET_VALUE **status**

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP,
RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM,
Every TMI Return Code

Example: Set NXP-ReadProtect Command to the tag 30-08-33-b2-dd-d9-01-40-35-05-00-00 with PUR-R:

PC -> Reader

52 46 45 01 5010 02 12 03 01 0C 300833b2ddd9014035050000 12345678 04 cs

dataLength	= 0x12
command	= 0x01 -> NXP-ReadProtect
tagIdCount	= 0x0C -> 12 Bytes
tagId	=30-08-33-b2-33-33-01-40-35-05-00-00
accessPassword	= 0x12345678

Reader -> PC 52 46 45 01 5010 02 01 03 00 04 cs

dataLength	= 0x01
status	= 0x00 -> RFE_RET_SUCCESS

6.51 Read-Multiple-From-Tag (50-20)

With this function data can be read from multiple memory banks of a tag with one command. The meaning of the data, that can be read, can be found in the manual of the tag. The memory banks are specified in the manual of the reader dependent of the type of tag.

Parameters: unsigned char **tagIdCount**, unsigned char **tagId**[tagIdCount]**,**
struct **MemBankInfo** {
unsigned char **memoryBank**, unsigned short **startAddress**,
unsigned long **accessPassword**, unsigned char **byteCount**
} [memBankCount]

Return Values: struct **MemBankData** {
unsigned char **byteCount,** unsigned char **data[byteCount]**
} [memBankCount]

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP,
RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM,
Every TMI Return Code

Example: Use the tag 30-08-33-b2-dd-d9-01-40-35-05-00-00 and read:
4 bytes from **memoryBank 0x01** at **address 8** with **password 01234567** and
16 bytes from **memoryBank 0x03** at **address 0** with **password 00000000**
PC -> Reader

52 46 45 01 5003 02 1D 03 0C 300833b2ddd9014035050000
01 0008 01234567 04
03 0000 00000000 10 04 cs
dataLength = 0x1E
tagIdCount = 0x0C -> 12 Bytes
tagId =30-08-33-b2-33-33-01-40-35-05-00-00
memoryBank = 0x01 -> second bank
startAddress = 0x0008
accessPassword = 0x0123456
bytesCount = 0x04 -> 4 Bytes
memoryBank = 0x03 -> fourth bank
startAddress = 0x0000
accessPassword = 0x00000000
bytesCount = 0x10 -> 16 Bytes

Reader -> PC

52 46 45 01 5003 02 1B 03 00 08 02120023A4884C15

10 55431813AB6841E36138AC3132183130 04 cs

dataLength	= 0x1B
status	= 0x00 -> RFE_RET_SUCCESS
bytesCount	= 0x08 -> 8 Bytes
data	= 0x02120023A4884C15
bytesCount	= 0x10 -> 16 Bytes
data	= 0x55431813AB6841E3 6138AC3132183130

6.52 Heartbeat-Interrupts (90-01)

The interrupt of the heartbeat is sent in a specified interval, if the heartbeat is turned on.

52 46 45 01 9001 02 **01** 03 00 04 cs

dataLength = 0x01 -> 1 Byte

status = 0x00 -> RFE_RET_SUCCESS

If a state heartbeat is selected, the heartbeat looks like follows.

52 46 45 01 9001 02 **02** 03 00 10 04 cs

dataLength = 0x02 -> 2 Byte

status = 0x00 -> RFE_RET_SUCCESS

state = 0x10 -> RFE_STATE_SCANNING

6.53 Inventory-Cyclic-Interrupt (90-02)

The interrupt to the cyclic inventory that can be started with the command 02-01-02 is sent from the reader to the host with exactly one TagInfo. The TagInfo is built up in the same way as it is described in the response of the single inventory. The length of the TagInfo is not included, because there will only be one TagInfo, so the length is not needed.

52 46 45 01 9002 02 **11** 03 01 0E 3000300833b23333014035050000 04 cs

dataLength = 0x11 -> 17 Bytes

TagInfo 1

StartByte-TagID = 0x01

ID Length = 0x0E -> 14 Bytes

TagID =30-00-30-08-33-b2-33-33-01-40-35-05-00-00

52 46 45 01 9002 02 **13** 03 01 0E 3000300833b23333014035050000 02 37 04 cs

dataLength = 0x13 -> 19 Bytes

TagInfo 1

StartByte-TagID = 0x01

ID Length = 0x0E -> 14 Bytes

TagID = 30-00-30-08-33-b2-33-33-01-40-35-05-00-00

StartByte-RSSI = 0x02

RSSI = 0x37

6.54 State-Changed-Interrupt (90-03)

The interrupt is sent every time the state changes.

52 46 45 01 9003 02 **01** 03 00 04 cs

dataLength = 0x01 -> 1 Byte

status = 0x00 -> RFE_STATE_IDLE

6.55 Status-Reg-Changed-Interrupt (90-04)

The interrupt is sent every time the reader detects an error.

52 46 45 01 9004 02 **08** 03 00 00 00 00 00 00 00 00 04 cs

dataLength = 0x04 -> 4 Byte

statusRegister = 0x00000000000000000000

6.56 Boot-Up Finished Interrupt (90-05)

The interrupt is sent when the system boot up finished.

The possible values are:

STARTING	**0x00**
FINISHED	**0x01**

52 46 45 01 9005 02 **01** 03 01 04 cs

dataLength = 0x01 -> 1 Byte

finished = 0x01 -> OK

6.57 Notification Interrupt (90-06)

The interrupt is sent every time a setting, for which the notification was activated, was changed.

52 46 45 01 9006 02 **04** 03 01 0D4094 04 cs

dataLength = 0x04 -> 4 Bytes

notification-id = 0x01 ->e.g. Frequency Change

frequency = 0x0D4094 -> 868500 kHz

6.58 Operation-Result-Interrupt (90-08)

The interrupt is sent every time a pending operation finished and contains the data of the operation result.

52 46 45 01 9008 02 **06** 03 23 00 03 02699A 04 cs

dataLength	= 0x06 -> 6 Bytes
pending-id	= 0x23 -> Pending ID 0x23
status	= 0x00 -> RFE_RET_SUCCESS
byteCount	= 0x03 -> 3 Bytes
result data	= 0x02669A

6.59 GPIO-Values-Changed-Interrupt (90-09)

The interrupt is sent every time a GPIO input pin's value is changed.

52 46 45 01 9004 02 **04** 03 00 00 98 0F 04 cs

dataLength	= 0x04 -> 4 Bytes
gpioValues	= 0x0000980F ->

Refer to the GPIO examples before:

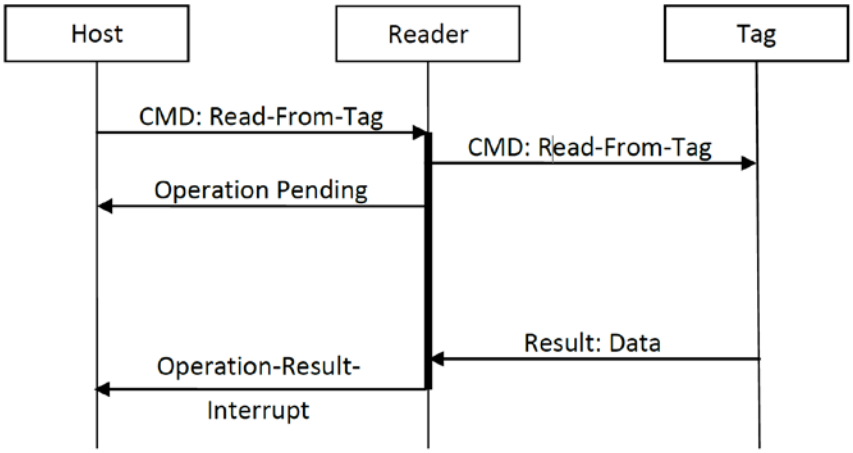
- Input Pins 15, 12 and 11 have a high level
- Output Pins 0 to 3 are set to a high level

7 Pending Results

Some operations, especially tag operations, can last for a quite long time. To not get in doubt if the reader did not get the command or is not responding any more, the return code RFE_RET_RESULT_PENDING was introduced. This return code should inform the application that the reader is working on the operation.

Example: Read-From-Tag command.

Message-Flow:



Example: Read 5 Byte of the tag 30-08-33-b2-dd-d9-01-40-35-05-00-00 at the memory bank 1 and the start address 0x12:

PC -> Reader

52 46 45 01 5003 02 15 03 0C 300833b2ddd9014035050000 01 0000 00000000 06 04 cs

Reader -> PC 52 46 45 01 5003 02 02 03 01 32 04 cs

dataLength	= 0x02
status	= 0x01 -> RFE_RET_OPERATION_PENDING
pending-id	= 0x32 -> Pending ID 0x32

Reader -> PC 52 46 45 01 9008 02 09 03 32 00 06 020023A4884C 04 cs

dataLength	= 0x09
pending-id	= 0x32 -> Pending ID 0x32
status	= 0x00 -> RFE_RET_SUCCESS
bytesCound	= 0x36 -> 6 Bytes
data	= 0x020023A4884C

8 Data Structures

The used data structures are collected in this chapter and shown in C syntax:

```
enum RFE_RET_VALUE
{
	RFE_RET_SUCCESS                     = 0x00,
	RFE_RET_RESULT_PENDING              = 0x01,
	RFE_RET_ERR_OP_NOT_SUPPORTED        = 0x50,
	RFE_RET_ERR_UNKOWN_ERR              = 0x51,
	RFE_RET_ERR_ON_EXEC_OP              = 0x52,
	RFE_RET_ERR_COULD_NOT_WRITE         = 0x53,
	RFE_RET_ERR_WRONG_PARAM_COUNT       = 0x54,
	RFE_RET_ERR_WRONG_PARAM             = 0x55,
	RFE_RET_TMI_TAG_UNREACHABLE         = 0xA0,
	RFE_RET_TMI_MEM_OVERRUN             = 0xA1,
	RFE_RET_TMI_MEM_LOCKED              = 0xA2,
	RFE_RET_TMI_INSUFFICIENT_POWER      = 0xA3,
	RFE_RET_TMI_WRONG_PASSWORD          = 0xA4
};

enum RFE_STATE_VALUE
{
	RFE_STATE_IDLE                      = 0x00,
	RFE_STATE_REBOOTING                 = 0x01,
	RFE_STATE_SCANNING                  = 0x10,
	RFE_STATE_WRITING                   = 0x11,
	RFE_STATE_READING                   = 0x12

};

enum HEARTBEAT_SIGNAL
{
	HEARTBEAT_OFF                       = 0x00,
	HEARTBEAT_ON                        = 0x01,
	HEARTBEAT_DUPLEX_ON                 = 0x02,
	HEARTBEAT_STATE_ON                  = 0x03,
	HEARTBEAT_DUPLEX_STATE_ON           = 0x04

};

enum ANTENNA_POWER
{
	ANTENNA_OFF                         = 0x00,
	ANTENNA_ON                          = 0x01
};

enum INVENTORY_MODE{
	INVENTORY_OFF                       = 0x00,
	INVENTORY_ON                        = 0x01
};
```